Inference Attack on Browsing History of Twitter Users using Public Click Analytics and Twitter Metadata

Jonghyuk Song, *Nonmember, IEEE,* Sangho Lee, *Member, IEEE,* and Jong Kim, *Member, IEEE*

Abstract—Twitter is a popular online social network service for sharing short messages (tweets) among friends. Its users frequently use URL shortening services that provide (i) a short alias of a long URL for sharing it via tweets and (ii) public click analytics of shortened URLs. The public click analytics is provided in an aggregated form to preserve the privacy of individual users. In this paper, we propose practical attack techniques inferring who clicks which shortened URLs on Twitter using the combination of public information: Twitter metadata and public click analytics. Unlike the conventional browser history stealing attacks, our attacks only demand publicly available information provided by Twitter and URL shortening services. Evaluation results show that our attack can compromise Twitter users' privacy with high accuracy.

Index Terms—URL Shortening Service, Twitter, Privacy leak, Inference

1 Introduction

Twitter is a popular online social network and microbloging service for exchanging messages (also known as *tweets*) among people, supported by a huge ecosystem. Twitter announces that it has over 140 million active users creating more than 340 million messages every day [26] and over one million registered applications built by more than 750,000 developers [25]. The third party applications include client applications for various platforms, such as Windows, Mac, iOS, and Android, and web-based applications such as URL shortening services, image-sharing services, and news feeds.

Among the third party services, URL shortening services which provide a short alias of a long URL is an essential service for Twitter users who want to share long URLs via tweets having length restriction. Twitter allows users to post up to 140-character tweets containing only texts. Therefore, when users want to share complicated information (e.g., news and multimedia), they should include a URL of a web page containing the information into a tweet. Since the length of the URL and associated texts may exceed 140 characters, Twitter users demand URL shortening services further reducing it.

Some URL shortening services (e.g., *bit.ly* and *goo.gl*) also provide shortened URLs' *public click analytics* consisting of the number of *clicks*, *countries*, *browsers*, and *referrers* of visitors. Although anyone can access the data to analyze visitor statistics, no one can extract specific information about individual visitors from the data because URL shortening services provide them as an *aggregated* form to protect the privacy of visitors from attackers.

• *The authors are with the Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea. E-mail: {freestar,sangho2,jkim}@postech.ac.kr.*

However, we detect a simple inference attack that can estimate individual visitors from the aggregated, public click analytics using *public metadata* provided by Twitter. First, we examine the metadata of client application and location because they can be correlated with those of public click analytics. For instance, if a user, Alice, updates her messages using the official Twitter client application for iPhone, "Twitter for iPhone" will be included in the source field of the corresponding metadata. Moreover, Alice may disclose on her profile page that she lives in the USA or activate the location service of a Twitter client application to automatically fill the location field in the metadata. Using this information, we can determine that Alice is an iPhone user who lives in the USA.

Next, we perform the simple inference attack on behalf of Alice's boyfriend, Bob, as follows. Bob first posts a tweet with a URL shortened by *goo.gl*. If Alice clicks on the shortened URL, *goo.gl* records {"country": "US", "platform": "iPhone", "referrer": "twitter.com", "browser": "Mobile"} in the click analytics of the shortened URL (details are in Sections 2 and 3). Otherwise, *goo.gl* records no information. Later, Bob retrieves the click analytics of the shortened URL to know whether Alice clicks on his URL. If the click analytics is unchanged or if its changes do not include information about the USA, iPhone, and twitter.com, he infers that Alice does not click on his URL. Otherwise, he infers that Alice click on his URL.

The main advantage of the preceding inference attack over the conventional browser history stealing attacks [3], [4], [8], [10], [14]–[17] is that *it only demands public information*. The conventional browser history stealing attacks rely on private information, such as Cascading Style Sheet (CSS) visited styles, browser cache, DNS cache, and latency. To collect such information, attackers should (i) prepare attack pages containing scripts/malware and lure target users for extracting the information from their web browsers or (ii) monitor DNS requests for measuring DNS lookup time of a target host. In other words, attackers should deceive or compromise target users or their networks to obtain the browsing history, which relies on strong assumption. In contrast, anyone can access the metadata of Twitter and the public click analytics of URL shortening services so that passive monitoring is enough for performing our attack.

In this paper, we propose novel attack methods for inferring whether a specific user clicked on certain shortened URLs on Twitter. As shown in the preceding simple inference attack, our attacks rely on the combination of publicly available information: click analytics from URL shortening services and metadata from Twitter. The goal of the attacks is to know which URLs are clicked on by target users. We introduce two different attack methods: (i) an attack to know who click on the URLs updated by target users and (ii) an attack to know which URLs are clicked on by target users. To perform the first attack, we find a number of Twitter users who frequently distribute shortened URLs, and investigate the click analytics of the distributed shortened URLs and the metadata of the followers of the Twitter users. To perform the second attack, we create *monitoring accounts* that monitor messages from all followings of target users to collect all shortened URLs that the target users may click on. We then monitor the click analytics of those shortened URLs and compare them with the metadata of the target user. Furthermore, we propose an advanced attack method to reduce attack overhead while increasing inference accuracy using the *time model* of target users, representing when the target users frequently use Twitter. Evaluation results show that our attacks can successfully infer the click information with high accuracy and low overhead.

We summarize the main contributions of this paper as follows:

- We propose novel attack techniques to determine whether a specific user clicks on certain shortened URLs on Twitter. To the best of our knowledge, this is the first study that infers URL visiting history on Twitter.
- We only use public information provided by URL shortening services and Twitter (i.e., click analytics and Twitter metadata). We determine whether a target user visits a shortened URL by correlating the publicly available information. Our approach does not need complicated techniques or assumptions such as script injection, phishing, malware intrusion, or DNS monitoring. All we need is publicly available information.
- We further decrease attack overhead while increasing accuracy by considering target users' time models. It can increase the practicality of our attacks so that we demand immediate countermeasures.

2 URL Shortening Services

In this section, we briefly introduce URL shortening services. The first notable URL shortening service is TinyURL, which was launched in 2002, and its success influences the development of many URL shortening services. URL shortening services reduce the length of URLs by providing short aliases of URLs to requesters and redirecting later visitors to the original URLs. The services are especially convenient for Twitter users, which imposes a limit on the length of a message. In the past, Twitter used TinyURL and *bit.ly* as the default URL shortening services. As of October 10, 2011, Twitter started using its own URL shortening service, *t.co*, to wrap all URLs in tweets in order to protect Twitter users from malicious URLs [24], [28].

Some URL shortening services also provide click analytics about each shortened URL. Whenever a user clicks on a shortened URL, information about the user is recorded in the corresponding click analytics. The click analytics is usually made public and anyone can access it. Among the services, we focus on *bit.ly* and *goo.gl* because they are broadly used and provide detailed information.

2.1 goo.gl

In December 2009, Google launched the Google URL Shortener at *goo.gl*. Its click analytics provides the following information about the visitors.

- Referrers
- Countries
- Browsers
- Platforms

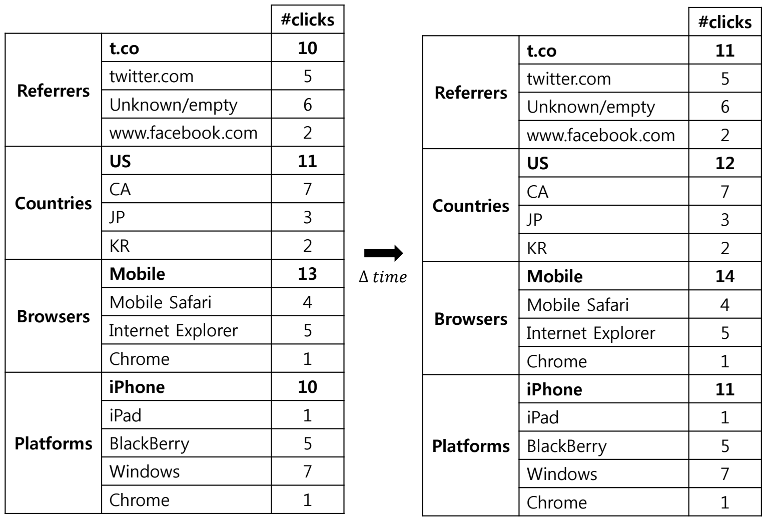
For example, let us assume a user uses a BlackBerry phone and lives in the USA. If he clicks on a shortened URL from *goo.gl* on Twitter, *gool.gl* records (i) *t.co* in the Referrers field, (ii) *Mobile Safari* in the Browsers field, (iii) *US* in the Countries field, and (iv) *BlackBerry* in the Platforms field of the corresponding click analytics. The reason why *t.co* is recorded in the Referrers field is that all links shared on Twitter are wrapped using *t.co* by Twitter from October 10, 2011.

2.2 bit.ly

Bitly company launched a URL shortening service *bit.ly* in 2008. Its click analytics provides the following information about visitors.

- Referrers
- Countries

Although *bit.ly* does not provide information about browsers and platforms, its Referrers field contains detailed information which can be utilized for inference. When a user clicks on a shortened URL on Twitter, *bit.ly* records the entire URL of the referrer site in the Referrers field, as "http://t.co/*****". In contrast, *goo.gl* records only "t.co" in the Referrers field. If we use the information provided by *bit.ly*, we can determine the exact URL of the tweet containing the clicked shortened URL. This information makes our inference attack possible even without having information about browsers and platforms.



		#clicks
Referrers	**t.co**	**10**
	twitter.com	5
	Unknown/empty	6
	www.facebook.com	2
Countries	**US**	**11**
	CA	7
	JP	3
	KR	2
Browsers	**Mobile**	**13**
	Mobile Safari	4
	Internet Explorer	5
	Chrome	1
Platforms	**iPhone**	**10**
	iPad	1
	BlackBerry	5
	Windows	7
	Chrome	1

$\xrightarrow[\Delta\, time]{}$

		#clicks
Referrers	**t.co**	**11**
	twitter.com	5
	Unknown/empty	6
	www.facebook.com	2
Countries	**US**	**12**
	CA	7
	JP	3
	KR	2
Browsers	**Mobile**	**14**
	Mobile Safari	4
	Internet Explorer	5
	Chrome	1
Platforms	**iPhone**	**11**
	iPad	1
	BlackBerry	5
	Windows	7
	Chrome	1

Fig. 1. The proposed system notices that there is a visitor using differences in the click analytics. We can infer that the information of the visitor is {"country": "US", "platform": "iPhone", "referrer": "t.co"} (*goo.gl* case)

3 Basics of Inference Attack

In this section, we introduce the basics of our inference attack. The basic idea of our attack is capturing *instant changes* in the public click analytics of shortened URLs by periodically monitoring it and matching the instant changes with the information about target users to infer whether our target users make the changes.

3.1 Periodic Monitoring and Matching

We periodically monitor click analytics of shortened URLs to observe its instant changes made by a new visitor. Whenever we notice that there is a new visitor, we match his or her information with each of our target users to know whether the new visitor is one of our target users. We can estimate information about visitors by checking the differences between the new and the old click analytics.

Fig. 1 shows an example of the process for obtaining the information about the visitor who clicks on a *goo.gl* URL. In this figure, we easily infer that the new visitor is an iPhone user lives in the USA because the numbers of clicks by "USA" and "iPhone" simultaneously increase.

In the periodic monitoring, determining the optimal query interval is important, which depends on the variety of the characteristics of followers. When there are some characteristics to be observed at the same time and their whole values change rapidly, the query interval should be short enough to catch a small change. As we have many followers, the slope becomes stiff so that we should have a short interval. In general, an interval should be decided by considering the slope of change in overall characteristics.

However, the periodic monitoring and matching have a limitation because Twitter does not officially provide personal information about users such as country, browsers, and platforms. We need to infer the information about target users by investigating their timeline and profile pages.

TABLE 1
An example of *bit.ly* click analytics. UID stands for a Twitter user ID and TID stands for a numerical ID of each tweet.

		#
Referrers	direct	2
	http://t.co/3slAb	8
	http://t.co/xInA4	4
	https://twitter.com/[UID]/status/[TID]	3
Countries	US	9
	KR	5
	ID	1
	CH	2

3.2 Referrers

We determine whether a new visitor comes from Twitter by using the changed referrer information of public click analytics. The click analytics of *goo.gl* only records the hostname of the referrer site. If a visitor comes from Twitter, "*t.co*" or "*twitter.com*" is recorded in the Referrers field. In most cases, "*t.co*" is recorded because all links shared on Twitter are automatically shortened to *t.co* links. *t.co* handles redirections by context and user agents so that the Referrer information varies according to the source of a click [27]. In some cases, "*twitter.com*" is recorded because some Twitter applications directly use original links instead of *t.co* links. Consequently, if the Referrers information of the visitor is "*t.co*" or "*twitter.com*", we regard the visitor as coming from Twitter.

In the case of *bit.ly*, we can analyze a shortened URL in detail because *bit.ly* records the entire URL of the referrer site in click analytics (Section 2.2 and Table 1). When a target user clicks on a *bit.ly* URL converted into a *t.co* URL, *bit.ly* records the entire *t.co* URL in the Referrers field. Referrer matching is considered successful when a *t.co* URL recorded in the click analytics is the same as the *t.co* URL of the target shortened URL.

TABLE 2
The examples of Browsers and Platforms corresponding to source

Source	Browsers	Platforms
Twitter for iPhone	Mobile	iPhone
Twitter for iPad	Mobile	iPad
Twitter for Android	Mobile Safari	Linux
Twitter for BlackBerry	Mobile Safari	BlackBerry

3.3 Country

We infer the country information of target users using the location field of their profile pages and compare it with the changed country information of public click analytics. In many cases, Twitter users fill in the location field with their city or place name. We can determine the user's country by searching GeoNames with the information in the location field of the user's Twitter profile. GeoNames returns the country code that corresponds to the search keywords. The country information provided by the click analytics is also a country code, so we have a successful country match if both country codes are the same.

However, the country matching has a limitation: it does not work when Twitter users leave the location field empty or fill in the location field with meaningless information (e.g., "earth" or "in your heart"). According to Hecht *et al.* [13], approximately 34% of Twitter users do not provide real location information. In the later experiments, we avoid such problems by only selecting target users who filled in valid location names in the location field. However, even without location information, our attacks are still possible with other information because the country information is not a major feature for conducting inference. Additionally, we can utilize recent studies inferring the location of Twitter users based on their posts [7], [13] for our attacks.

3.4 Browsers and Platforms

When our target users click on a shortened URL provided by *goo.gl*, we use the information about the user's browser and platform to increase the inference accuracy. Although Twitter does not provide information of this nature about its users, it does record the name of the application that was used to post a tweet. For example, when someone posts a tweet using the official Twitter client application for the iPhone, the information "Twitter for iPhone" is recorded in the source field of the tweet, which enables us to use this information to infer the browser and platform that were used. Table 2 shows examples of source values corresponding to different browsers and platforms.

We should consider applications supporting multiple platforms, such as TweetDeck, which is a multi-platform application that is supported by the iOS, Android, Windows, and Mac OS X operating systems. A target user, who uses a multi-platform application, should be regarded as using all the platforms that support the application.

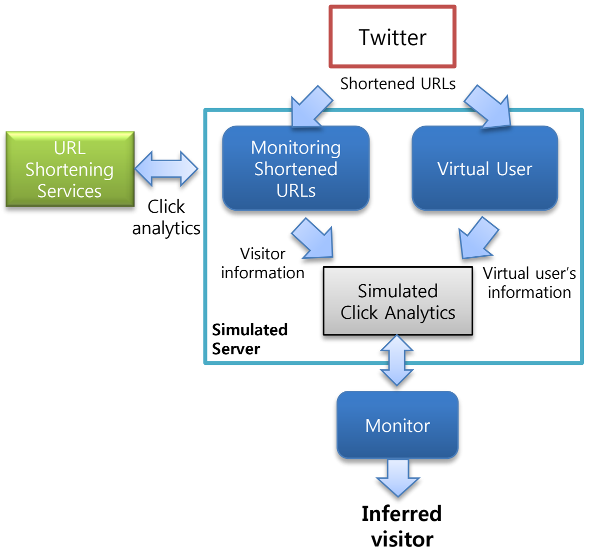


Fig. 2. Overall architecture of the attack in the simulated environment

4 Inference Attack in the Simulated Environment

In this section, we evaluate our inference attack in the simulated environment to correctly identify its accuracy. The definite ways to exactly evaluate our attacks are (i) asking the target users whether they really visited the shortened URLs or (ii) monitoring their browsing activities by using logging software. But, both approaches are restrictive because we cannot survey all of them or require them to install the logging software. Therefore, we construct a simulated environment supported by real click analytics to perform our attacks in almost real environments.

Fig. 2 shows the overall architecture of the attack in the simulated environment. In this experiment, we use *virtual users* instead of Twitter users in the real world. The attack system tries to differentiate shortened URLs clicked on by the virtual users from shortened URLs clicked on by real Twitter users. The processes involved in this attack system are as follows:

1) The monitor module in the simulated server periodically monitors the click analytics of the shortened URLs posted by some Twitter users.
2) The monitor module extracts real visitor information from the click analytics according to its changes and records it in the simulated click analytics.
3) The virtual user module in the simulated server *stochastically* adds the information about a virtual user to the simulated click analytics to simulate the click of the virtual user.
4) The inference system periodically checks the changes in the simulated click analytics and extracts the information about a new user from it.
5) The inference system compares the information about the new user and the information about the virtual user. If they are matched, we infer that the shortened URL is clicked on by the virtual user.

We prepare the followings for the experiment. First, we

TABLE 3
Twitter users used in the simulated experiment

# of followers	*goo.gl*	*bit.ly*
100–1k	3	3
1k–10k	10	5
10k–100k	9	13
100k–1M	7	6
total	29	27

find 56 Twitter users who frequently post *goo.gl* or *bit.ly* shortened URLs and have a diverse number of followers between 100 and 1 million (Table 3). The simulated server monitors shortened URLs that they update. Second, we empirically test various probability of whether virtual users click on shortened URLs. We finally select a probability of 0.7 because it corresponds to the reduction ratio (Section 5). Third, we prepare 24 virtual users having different country and platform information. We cannot test all types of Twitter users because they are too diverse, so we restrict the number of user types for our experiment. We first select six different countries: United States (US), Great Britain (GB), Brazil (BR), Japan (JP), Italy (IT), and Rwanda (RW). The first five are among the top 20 countries with the largest number of Twitter accounts [22]. We also consider Rwanda to learn the effectiveness of the system when the target user lives in a country with only a few Twitter users. We then select four different smart phone platforms: iPhone, Android, BlackBerry, and Windows Phone. Combinations of the six countries and the four platforms give us 24 different types of virtual users for the experiments.

In this experiment, the relatively small number of Twitter users posting shortened URLs, 56, is not important because the inference accuracy of our system mainly depends on the number of followers of the posting users, not the number of posting users. Whether our system can correctly recognize the clicks of virtual users depends on the uniqueness of the virtual users against other followers of posting users, so preparing diverse types of virtual users and posting users having a diverse number of followers are the most important to evaluate our system. The 56 posting users we found have a diverse number of followers, so they are enough to be used for evaluating our system.

4.1 Data Collection

We collected data by crawling the click analytics of the shortened URLs, using the API methods offered by *goo.gl* and *bit.ly*. *goo.gl* APIs have a rate limit of 1,000,000 queries per day. Similarly, *bit.ly* allows users to create no more than five concurrent connections from one IP address. *bit.ly* also enforces per-hour limits, per-minute limits, and per-IP rate limits for each API method. However, *bit.ly* does not publish the exact number of allowed requests on each limit. In all, we monitored 31,525 *goo.gl* URLs and 24,144 *bit.ly* URLs from September to October 2012. Those shortened URLs posted by 56 Twitter users (Table 3).

TABLE 4
Confusion matrix

		Actual value Click	Non click
Prediction Value	Click	True positive	False positive
	Non click	False negative	True negative

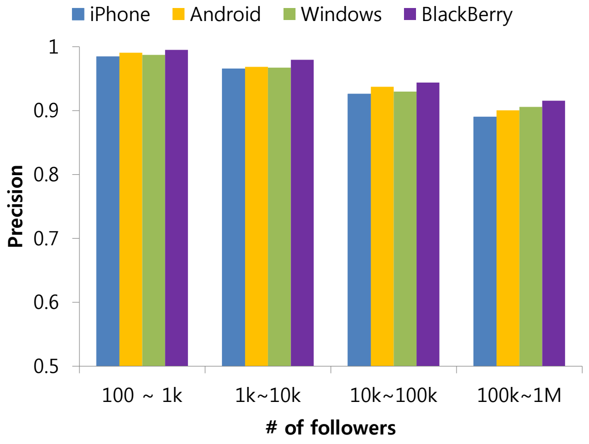


Fig. 3. The precision of *goo.gl* URLs in terms of platforms. X axis means the number of followers of the updating users.

4.2 Evaluation

In this experiment, we use two metrics to evaluate our attacks: precision and false positive rate (FPR) defined as follows:

$$\textbf{Precision} = \frac{\textbf{True positive}}{\textbf{True positive} + \textbf{False positive}},$$
$$\textbf{FPR} = \frac{\textbf{False positive}}{\textbf{False positive} + \textbf{True negative}}.$$

We do not consider the false negative rate because it is always one in the simulated environment. False negatives occur when a virtual user clicks on the shortened URL, but the system infers that the virtual user does not click on the URL (Table 4). In the real world, the system is occasionally unable to obtain all information about the target user if he uses several platforms and browsers. In the simulated environment, however, the system knows all information about the virtual users, and the information is unchanged during the entire experiment. Therefore, the system always knows what URLs are clicked on by the virtual user by monitoring the simulated click analytics.

In contrast, false positives are possible because some Twitter users have the same information as the virtual users. If such users click on the shortened URLs monitored by our attack system, we obtain a false positive result.

4.2.1 *goo.gl*

We first evaluate our attack system on *goo.gl* URLs. We create a Twitter account and follow 29 Twitter users who frequently post *goo.gl* URLs. We then monitor their URLs in the simulated environment while adding virtual clicks.

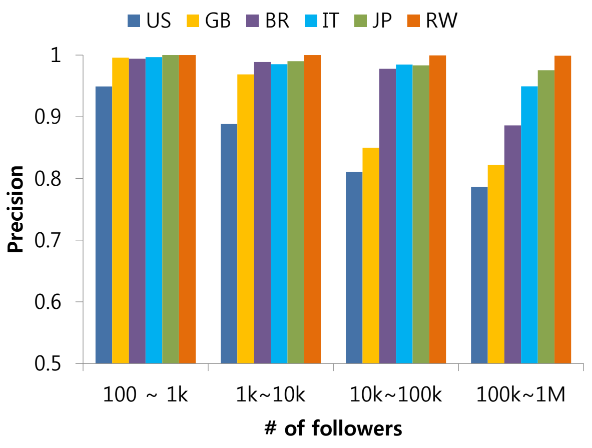


Fig. 4. The precision of *goo.gl* URLs in terms of country. X axis means the number of followers of the updating users.

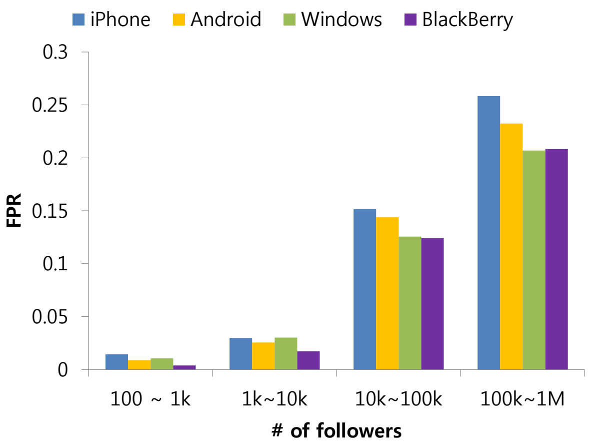


Fig. 5. The FPR of *goo.gl* URLs in terms of platforms. X axis means the number of followers of the updating users.

We observe that the accuracy of our system decreases according to the number of the followers of the posting users. As the number of followers increases, the probability that some of the followers and the virtual users have the same features (e.g., country and platform) increases. Consequently, our system may incorrectly guess virtual users because it misjudges the followers as the virtual user.

We group the posting users based on the number of their followers to identify how the number of followers affects the experiment results. First, Fig. 3 shows the precision of each platform according to the number of followers. We expect to see low precision with iPhone and Android users due to the large number of users on those platforms. As expected, the results show that our system has the lowest precision with iPhone users; however, the differences among the platforms are minimal. Average precisions of each platform are as follows: iPhone is 0.94, Android is 0.95, Windows Phone is 0.95, and BlackBerry is 0.96.

Second, Fig. 4 shows the precision of each country according to the number of followers. We have the lowest precision with US users because they comprise a large percentage of the total number of Twitter accounts [22]. Average precisions of each country are as follows: US is 0.85, GB is 0.90, BR is 0.96, IT is 0.97, JP is 0.98, and RW is 0.99.

Both results show that the precision decreases as the number of followers increases. The average precision is 0.99 when the number of followers is less than 1,000, but the average precision is 0.90 when the number of followers is greater than 100,000.

We also check FPR in the same settings (Figs. 5 and 6). As the figures show, FPR increases according to the number of followers of the posting users, and US and iPhone users have higher FPR than others. The average FPR is 0.1.

4.2.2 bit.ly

Second, we evaluate our attack system on *bit.ly* URLs. We create another monitoring account following 27 Twitter users who regularly update *bit.ly* URLs. Figs. 7 and 8 show the precision and FPR in terms of countries in the *bit.ly* cases, which are similar to those of *goo.gl*. US users also have lower precision and higher FPR than others.

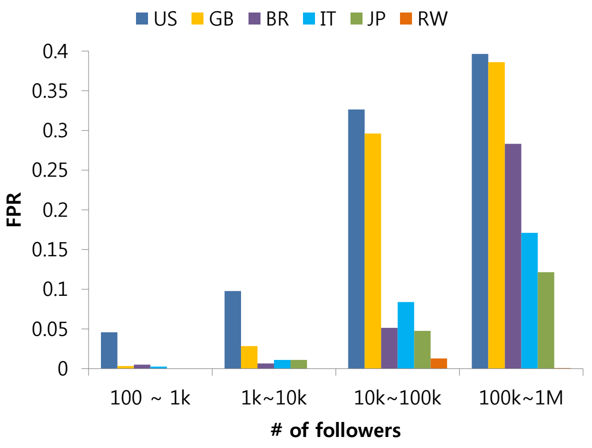


Fig. 6. The FPR of *goo.gl* URLs in terms of country. X axis means the number of followers of the updating users.

The overall accuracy of our attack system is lower with *bit.ly* than that with *goo.gl*, because *goo.gl* offers four types of information in the click analytics, whereas *bit.ly* offers only two types of information, namely, the Referrers and the Countries, as mentioned in Section 2.2. The average precision is 0.87 and the average FPR is 0.16.

4.2.3 Discussion

We observe that the accuracy of our attacks heavily depends on the number of Twitter users sharing the same information as the target user: following the same Twitter users, living in the same country, and using the same platform. If no other user has the same information as the target user, our system can correctly infer the target user regardless of the number of the posting user's followers. For example, the attack system can correctly infer most of the URLs clicked on by the virtual Rwanda users regardless of the number of followers and platforms. In contrast, the system has the lowest accuracy if the target user lives in the US and uses an iPhone: the precision of 0.81 and the FPR of 0.28.

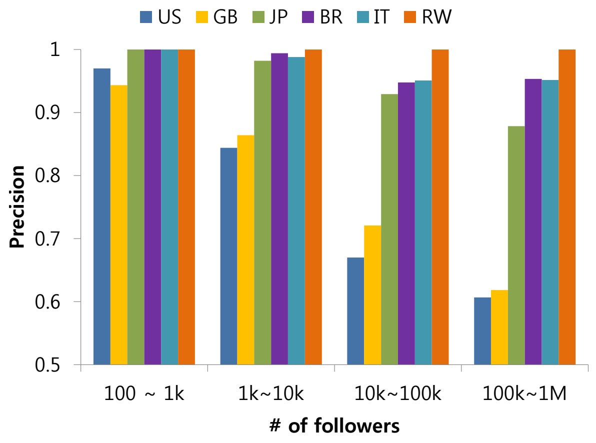


Fig. 7. The precision of *bit.ly* URLs in terms of country. X axis means the number of followers of the updating users.

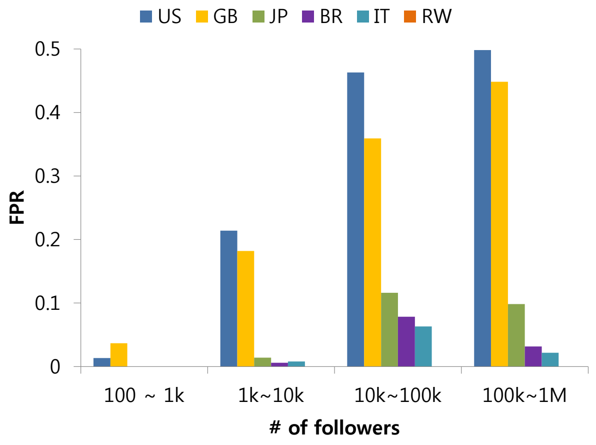


Fig. 8. The FPR of *bit.ly* URLs in terms of country. X axis means the number of followers of the updating users.

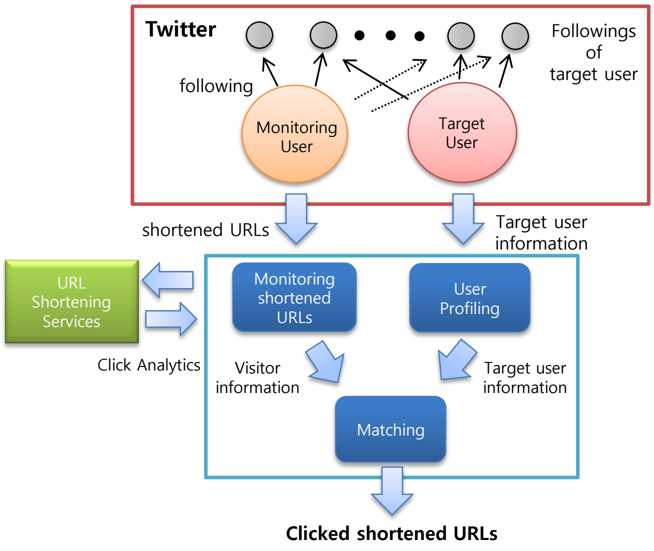


Fig. 9. Overall architecture of the attack I

5 Inference Attacks in the Real World

In this section, we introduce the inference attacks in the real world. We consider two inference attack scenarios: (i) inferring which URLs target users click on and (ii) inferring who clicks on the URLs updated by target users.

5.1 Attack I: Inferring Visited Shortened URLs by Target Users

In Attack I, our attack system identifies whether a target user clicks on shortened URLs posted by his or her followings. The results of this attack is a set of URLs that the target user may click on. The procedures of this attack are as follows:

1) The attack system chooses a target Twitter user and extracts his or her information from Twitter.
2) The system monitors the click analytics of all shortened URLs posted by the followings of the target user.
3) When the system notices changes in the click analytics, which indicate a new visitor to the shortened URL, the system extracts the visitor's information from the click analytics.
4) The system compares the information about the visitor with the known information the target user. If both pieces of information match, it infers that the target user clicks on the shortened URL.

Fig. 9 shows the overall architecture consisting of three modules: *profiling module*, *monitoring module*, and *matching module*. First, the profiling module obtains the information of the target user from the target user's profile and timeline, as mentioned in Sections 3.3 and 3.4. Second, the monitoring module extracts the shortened URLs from the tweets posted by the followings of the target user and monitors the changes in the click analytics of the shortened URLs. We create a Twitter user (*monitoring user*) who follows all the followings of the target user in order to access all tweets that the target user may view. Lastly, the matching module compares the information about the new visitor with the information about the target user when the monitoring module notices the changes in the click analytics. If the matching module infers that the new visitor is the target user, it includes the corresponding shortened URL in a *candidate URL set*.

The final candidate URL set may not accurate due to two reasons. First, it may contain URLs clicked on by other Twitter users who have the same features as the target user (false positives). There are many Twitter users who receive the same shortened URLs seen by the target user. Some of them may share the same information as the target user and click on a shortened URL that is being monitored by our system. In such cases, the system may conclude that the shortened URL clicked on by the target user.

Second, the final candidate set may not include a shortened URL clicked on by the target user when the target user clicks on the shortened URL in different country and/or platform (false negatives). For example, when a target user

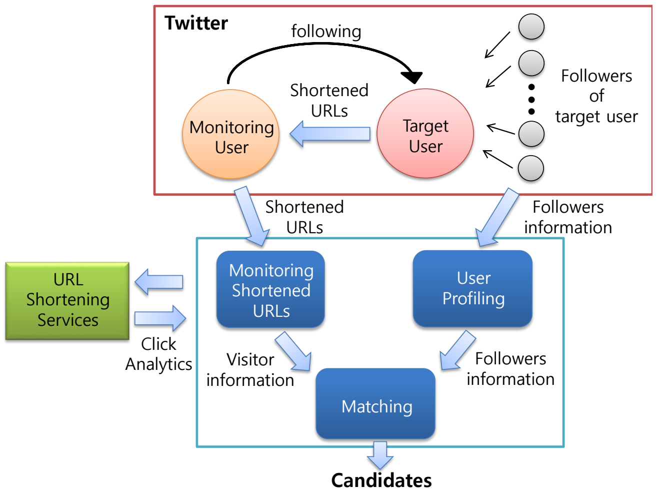


Fig. 10. Overall architecture of the attack II

typically uses an iPhone in the USA, our attack system only considers changes in click analytics involving the iPhone and the USA. However, it is possible that the target user changes his smart phone or travels to a foreign country. If he clicks on the shortened URLs in such environments, our system cannot notice the click events. We reduce the false negatives by promptly adding the new environment of a target user into his profile information. Therefore, the false negatives only temporarily occur and do not much affect the overall results.

5.2 Attack II: Inferring Visitors of Shortened URLs

In Attack II, our attack system identifies who clicks on shortened URLs uploaded by a target user. We first find a target Twitter user who periodically updates *bit.ly* or *goo.gl* shortened URLs and identify candidates who click on the shortened URLs updated by the target user. The procedures of this attack are as follows:

1) The attack system selects a target Twitter user who periodically updates shortened URLs.
2) The system monitors the click analytics of shortened URLs updated by the target user.
3) When the system notices that there is a visitor of the shortened URL, it extracts the visitor information from the click analytics.
4) The system compares the information about the visitor with the known information of the followers of the target users.

Fig. 10 shows the overall architecture consisting of the three modules (monitoring, profiling, and matching modules), which are similar to those of Attack I. First, the monitoring module receives the shortened URLs updated by the target user and extracts their click analytics. Unlike Attack I, the monitoring user of Attack II just follows the target user to receive his or her shortened URLs. Next, the profiling module collects the information of the followers of the target user when the monitoring module notices a new visitor. We collect the information of the followers on demand to obtain up-to-date information of the old followers and information of new followers. Lastly, the matching module compares the information of the new visitor with the information of the followers. If the matching module infers that the new visitor is a follower of the target user, it includes the user in a *candidate user set*.

The final candidate user set may contains wrong candidate users because the target user has a number of followers who share the same features. For instance, if an iPhone user who is staying at the USA clicks on the shortened URL, some followers of the target user using iPhone and staying at USA can be candidate users. However, in most cases, the number of candidates is smaller than the number of followers of the target user, so our attack is better than a random guessing attack that considers all followers as candidates. Moreover, if some followers of the target user have unique features (e.g., lives in Rwanda), we can exactly infer the shortened URLs clicked on by them.

5.3 Target User Selection

We choose target users of Attack I according to the following four criteria. First, we choose target users who have identifiable and public profile information. Also, we select users who use well-known applications, such as the official Twitter applications for smart phones. If we select users who use unpopular applications, the inference system will identify the target user with a high accuracy, but it gives biased results. Therefore, we conduct experiments with common users to prove that our system has good performance in most cases.

Second, the target users should follow some Twitter users frequently posting *bit.ly* or *goo.gl* URLs because we want to obtain enough experimental results. If no shortened URL appears in the timeline of the target user, we cannot perform our attacks.

Third, the target users must actively use Twitter. If we select an inactivate user as a target user, we cannot obtain enough experimental data. Our ideal target users are Twitter users who frequently check their timeline and click on URLs on their timeline.

Lastly, the target users need to post or retweet a tweet including the shortened URLs that he clicks on. We assume that we successfully infer the clicks on the shortened URLs if the target user posts a tweet with the shortened URL in a candidate URL set. In order to find qualified target users for the experiments, we manually search *goo.gl* or *bit.ly* strings on Twitter and review their timeline. However, the preceding criteria are used only to obtain enough experimental data and to conduct evaluation. They are not strongly related to the accuracy of our attacks. Any Twitter user can be a target user.

In Attack II, we choose target users who frequently update shortened URLs for obtaining enough experimental data. Twitter accounts of news, advertisements, and celebrities are the best target users because they periodically update shortened URLs and have a large number of followers.

We manually find target users by searching *goo.gl* or *bit.ly* strings on Twitter.

5.4 Data Collection

We crawl Twitter data using two sets of Twitter API methods: Streaming APIs and REST APIs. The Streaming APIs allow us to monitor target users in real time. We use the REST APIs for crawling profile pages, timelines, followers, and followings. The REST APIs, however, have a rate limit: a host is permitted 150 requests per hour. In order to overcome the rate limit, we change the IP address of the crawling servers when the servers exceed the rate limit. We used 10 servers and 100 IP addresses to crawl Twitter data. We also collect the click analytics of the shortened URLs as mentioned in Section 4.1.

We collected 219,459 user's profiles, 197,879 user's timelines, 183,176 user's favorites and monitored 3,106 goo.gl URLs and 26,532 bit.ly URLs. The collection lasted for about two month from March 2012 to April 2012.

5.5 Evaluation

As mentioned in Section 4, it is difficult to correctly evaluate our attack system because we cannot obtain click logs of target users. Thus, in this section, we use tweets and favorites of target users to evaluate our attack system instead of click logs. We assume that Twitter users include URLs in their tweets and favorite tweets with URLs only when they previously visit the URLs. Based on the assumption, we check whether a target user includes the URL inferred as visited in his (re)tweets or favorites it in the near future to validate the correctness of our inference. To clarify, suppose that our system infers that a Twitter user A visits a shortened URL U. We collect the timeline and the favorites of the user A and check whether a tweet containing the shortened URL U exists. If we find the shortened URL U in the timeline or favorites of the user A, we are certain that the system successfully infers shortened URLs visited by the user A.

5.5.1 *Attack I*

We evaluate the accuracy of Attack I by introducing the following three metrics:

$$\mathbf{P1} = \frac{|\mathbf{U} \cap \mathbf{RT}|}{|\mathbf{U}|},$$
$$\mathbf{P2} = \frac{|\mathbf{C}_{urls} \cap \mathbf{RT}|}{|\mathbf{C}_{urls}|},$$
$$\mathbf{P3} = \frac{|\mathbf{N}_{urls} \cap \mathbf{RT}|}{|\mathbf{N}_{urls}|},$$

where $\mathbf{U}$ is a set of all shortened URLs posted by followings of the target user, $\mathbf{C}_{urls}$ is a candidate URL set including shortened URLs inferred as visited by the target user, $\mathbf{N}_{urls}$ is a non-candidate URL set including shortened URLs inferred as unvisited, and $\mathbf{RT}$ is a retweeted URL set including shortened URLs which are in the target user's timeline or favorite lists, satisfying the following conditions:

$$\mathbf{C}_{urls} \subseteq \mathbf{U}, \mathbf{N}_{urls} \subseteq \mathbf{U}, \mathbf{RT} \subseteq \mathbf{U},$$
$$\mathbf{C}_{urls} \cup \mathbf{N}_{urls} = \mathbf{U}, \mathbf{C}_{urls} \cap \mathbf{N}_{urls} = \phi.$$

Therefore, **P1** indicates the fraction of retweeted shortened URLs, **P2** indicates the fraction of retweeted candidate URLs, and **P3** indicates retweeted non-candidate URLs.

The final values of the three metrics are as follows: **P1** is 0.032, **P2** is 0.048, and **P3** is 0.003. **P2** is 1.5 times higher than **P1** and 16 times higher than **P3**, which implies that we can successfully categorize all shortened URLs into a set of visited URLs and a set of unvisited URLs. The target users normally post tweets containing shortened URLs included in the candidate URLs set and rarely post tweets with shortened URLs outside the candidate URLs set. According to boyd *et al.* [2], approximately three percent of tweets are likely to be retweeted. This percentage is similar to our calculation of **P1**, 0.032; therefore, we conclude that the value of **P1** is trustworthy.

Next, we introduce two different metrics to view the results from a different angle:

$$\mathbf{P4} = \frac{|\mathbf{C}_{urls} \cap \mathbf{RT}|}{|\mathbf{RT}|},$$
$$\mathbf{P5} = \frac{|\mathbf{N}_{urls} \cap \mathbf{RT}|}{|\mathbf{RT}|}.$$

P4 indicates the fraction of retweeted URLs included in the candidate URL set and **P5** indicates the fraction of retweeted URLs included in the non-candidate URL set.

The final values of the two metrics are as follows: **P4** is 0.952 and **P5** is 0.048. We observe that **P4** is much higher than **P5**, which implies that most of the shortened URLs that are in the timeline or favorites of the target users are inferred as candidate URLs. Therefore, we conclude that a shortened URL is highly likely to be retweeted or favorited by the target user if it is included in the candidate set.

The problem of the metrics **P4** and **P5** is that we can simply achieve the best results when we include all the shortened URLs in the candidate URL set. To show that we correctly infer the candidate URL set, we compute the reduction ratio **RR**, which represents how much we reduce the number of candidate URLs from the number of all shortened URLs posted by the followings of the target user.

RR is computed as follows:

$$\mathbf{RR} = \frac{|\mathbf{C}_{urls}|}{|\mathbf{U}|}.$$

Table 5 shows the reduction ratios of shortened URLs from the two URL shortening services: the average value of the reduction ratio is 0.669. Since the reduction ratio is quite smaller than **P4**, we can conclude that our attack system intelligently reduces the candidate set. We have a larger number of *bit.ly* shortened URLs than that of *goo.gl* shortened URLs because the number of *bit.ly* shortened URLs is fairly larger than that of *goo.gl* on Twitter.

TABLE 5
The monitored shortened URLs and **RR** for each URL shortening services in Attack I.

	# of shortened URLs	**RR**
goo.gl	2,278	0.584
bit.ly	25,816	0.674
Total	28,094	0.669

In addition, the value of **RR** depends on click tendency of the target users. If the target users click on all of the shortened URLs in **U**, **RR** becomes 1 so that high **RR** does not always indicate that the system is performing poorly.

5.5.2 Attack II

We evaluate the accuracy of Attack II by introducing the following two metrics:

$$\mathbf{P6} = \frac{|\mathbf{RT}_C|}{|\mathbf{RT}_F|},$$
$$\mathbf{P7} = \frac{|\mathbf{RT}_N|}{|\mathbf{RT}_F|},$$

where $\mathbf{RT}_F$ is a set of shortened URLs of a target user retweeted or favorited by his or her followers, $\mathbf{RT}_C$ is a set of shortened URLs retweeted or favorited by the candidate users, and $\mathbf{RT}_N$ is a set of shortened URLs retweeted by the non-candidate users, satisfying the following conditions:

$$\mathbf{RT}_C \subseteq \mathbf{RT}_F, \mathbf{RT}_N \subseteq \mathbf{RT}_F,$$
$$\mathbf{RT}_C \cup \mathbf{RT}_N = \mathbf{RT}_F, \mathbf{RT}_C \cap \mathbf{RT}_N = \phi.$$

Therefore, **P6** indicates the fraction of retweeted URLs by candidate users and **P7** indicates the fraction of retweeted URLs by non-candidate users.

If the attack system correctly infers candidate users, **P6** should be higher than **P7**. In this experiment, we monitor 20 target Twitter users and each target user has a number of followers between several thousands to tens of thousands. We compute **P6** and **P7** of each target user: the average value of **P6** is 0.86 and the average value of **P7** is 0.14. As we expected, **P6** is much higher than **P7** so that our attack system can successfully distinguish candidate users from non-candidate users.

Again, we should consider the reduction ratio **RR** representing how much we reduce the number of candidate users from the number of the target user's followers as follows:

$$\mathbf{RR} = \frac{|\mathbf{RT}_C|}{|\mathbf{followers}|}.$$

Table 6 shows the results for each URL shortening services: the average value of **RR** is 0.474 which is fairly smaller than **P6**. Thus, our attack system infers about half of the target user's followers as the candidate users on average. This result is much better than a random guessing attack because the random guessing attack should consider all followers as candidate users. The reduction ratio in the *goo.gl* case is lower than in the *bit.ly* case because *goo.gl* provides more detailed information than *bit.ly*.

TABLE 6
The monitored shortened URLs and **RR** for each URL shortening services in Attack II.

	# of shortened URLs	**RR**
goo.gl	827	0.474
bit.ly	716	0.768
Total	1,544	0.669

TABLE 7
Retweeted tweets containing a shortened URL according to the number of followers.

# of followers	# of tweets	# of retweeted tweets
100 – 1k	22,442 (0.57)	833 (0.04)
1k – 10k	13,224 (0.34)	1,216 (0.09)
10k – 100k	2,930 (0.08)	882 (0.30)
100k – 1000k	453 (0.01)	277 (0.61)
Total	39,049	3,208 (0.08)

5.5.3 Uniqueness of a shortened URL

A tweet containing a shortened URL can be read by users who are not followers of the user who posted the tweet, via retweets or other channels, so non-followers may click on the shortened URL. Therefore, when inferring candidate users, our inference system needs to consider not only the followers of a user but also those who can read the users tweets. In Attack II, however, we conduct the experiment under the assumption that only the followers of a user will click on a shortened URL, even though non-followers may affect the experimental results. Specifically, $\mathbf{RT}_C$ may contain a shortened URL clicked by a non-follower when the non-follower has the same features as a follower of the user and the follower retweets the shortened URL without clicking on it.

However, the effect of non-followers on the experimental results is negligible because the number of clicks by the non-followers is small. To demonstrate this point, we determine the number of retweeted tweets and the number of users who posted the same shortened URL. We crawled tweets that were older than one day because we wanted to collect tweets that had enough time to be retweeted. To avoid bias in the results, we randomly collected tweets that contained a shortened URL by using a Twitter search API with keywords *goo.gl* and *bit.ly*. We collected 17,227 tweets that contained goo.gl URLs and 21,822 tweets that contained bit.ly URLs, from June 2014 to July 2014.

First, we determine the number of retweeted tweets that contained shortened URLs. As shown in Table 7, overall, 8% of the collected tweets were retweeted. If the users who have more than 100,000 followers posted tweets, 61% of those tweets were retweeted. However, these tweets account for only 1% of the collected tweets. If the users who have less than 1,000 followers posted tweets, only 4% of those tweets were retweeted, but they account for 57% of the collected tweets. Consequently, we conclude that most tweets are not retweeted.

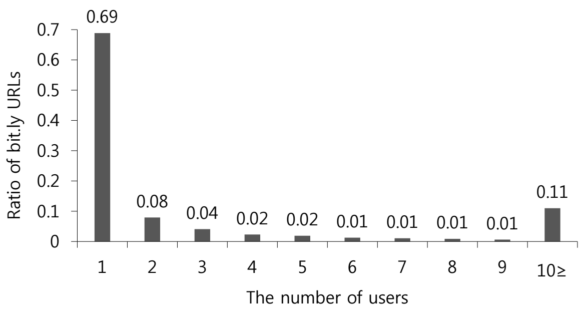


Fig. 11. The ratio of bit.ly URLs according to the number of users who post the same bit.ly URL.

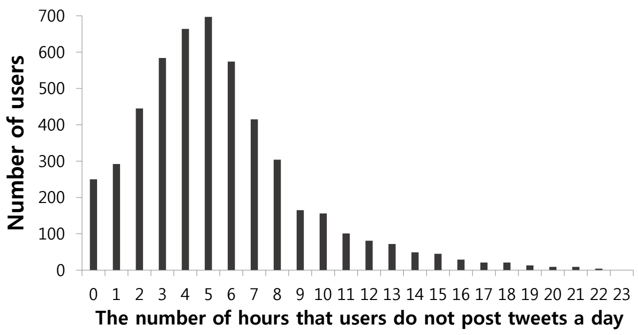


Fig. 12. The distribution of the number of hours that users do not post any tweets

Next, we determine the number of users who post the same shortened URL, while excluding retweets. Unfortunately, we can not know the exact number of tweets that contained a specific URL because the Twitter search API returns only a small number of tweets containing search keywords. Instead, we use the click analytics of the collected bit.ly URLs to inspect the number of users who post the same bit.ly URL on Twitter. Fig. 11 shows that approximately 69% of the bit.ly URLs are posted by a single user on Twitter. The next highest proportion is only 8%, which represents the proportion of bit.ly URLs posted by two users.

We can conclude that the effect of non-followers on the experimental results in Section 5.5.2 is negligible because most tweets are not retweeted and most shortened URLs are posted by a single user. Therefore, the evaluation results of Attack II are reliable.

6 Advanced Inference Attack

In this section, we introduce an advanced inference attack that decreases attack overhead while increasing inference accuracy by considering *when target users frequently use Twitter*. If we are assured that a target user does not use Twitter during some time periods (e.g., in the middle of night), we do not need to collect and inspect click logs recorded in the time periods. This exclusion not only

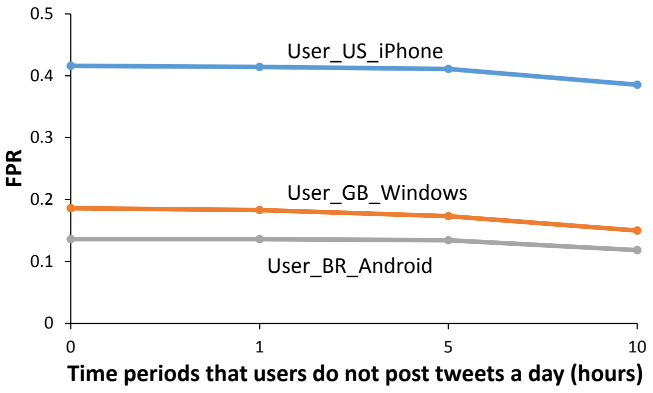


Fig. 13. The FPR of *goo.gl* URLs in the advanced inference attack.

reduces attack overhead (e.g., OAuth tokens and system uptime), but also increases inference accuracy (e.g., reducing false positives). We first analyze tweet history of Twitter users to build time models, and then evaluate the advanced attack by conducting experiments with virtual users following the time models.

6.1 Time Model

We first build time models that represent when Twitter users use Twitter by using their *tweet history* that records when they post tweets. We anticipate that the difference between a time model and the tweet history is small because we mainly focus on *heavy Twitter users* who frequently post tweets during almost all their waking hours. We use time models to configure time-based behaviors of virtual users in a simulated environment.

The procedure for obtaining time models is as follows. First, we randomly choose 5,000 Twitter users from Twitter public timeline and collect each of their last 30-days tweets. Next, for each user, we divide a day into 24 hour buckets and investigate the number of *idle* hour buckets that the user never posts tweets during the last 30 days. For example, if a user never posts tweets from 2:00 a.m. to 5:00 a.m. and from 10:00 p.m. to 11:00 p.m. during all of the last 30 days, the number of his idle hour buckets is four. Fig. 12 shows the distribution of the number of idle hour buckets. On average, the Twitter users we analyzed do not post tweets five hours a day. Interestingly, approximately five percent of the Twitter users post tweets in all day; we anticipate that they are bot accounts. Lastly, we select three time models for experiments: (i) posting no tweet five hours a day, (ii) posting no tweet one hour a day, and (iii) posting no tweet ten hours a day. The first one is the most common time model whereas the remaining two models are extreme cases.

6.2 Experiment Setup and Data Collection

We establish the simulated environment of Section 4 while varying the behaviors of virtual users according to the

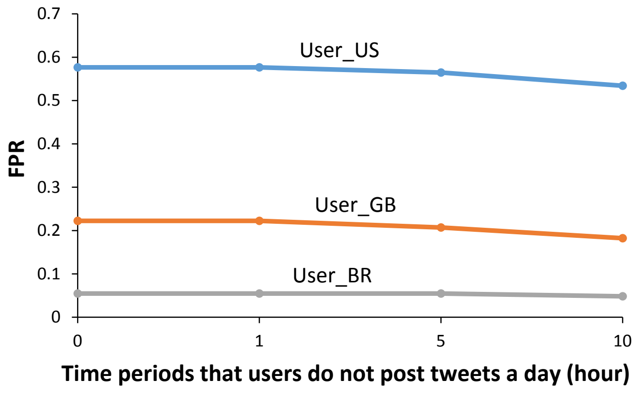


Fig. 14. The FPR of *bit.ly* URLs in the advanced inference attack.

time models explained in Section 6.1. We generate four variations of each virtual user in Section 4: a user using Twitter 24 hours a day, a user using Twitter 23 hours a day, a user using Twitter 19 hours a day, and a user using Twitter 14 hours a day. For simplicity, we set up the idle hour buckets of each virtual user are continuous and do not vary day-to-day. Such settings are acceptable because whether the idle hour buckets are continuous or not does not affect much the experiment results. By monitoring a target user, we can exclude logs of non-active hours, regardless of whether these non-active hours are continuous or not. The important thing is how many hours the target user is non-active.

We monitored 2,570 *goo.gl* URLs and 1,550 *bit.ly* URLs from November 8 to November 29, 2013 in the same way as the previous experiments.

6.3 Evaluation

We evaluate the results of our experiment according to inference accuracy and attack overhead. We observe slightly increased precision and decreased FPR when we consider time models. Fig. 13 and Fig. 14 show that FPR decreases as virtual users rarely click on. If the X axis value is zero, the users click the shortened URLs all the time that the previous virtual users did. If the X axis value is 10, the users did not click the shortened URLs during 10 hours a day. The inference system excludes the click logs in the time periods that the users do not click. Therefore, the inference system can reduce false positive clicks when the time periods that users do not click are long. From the results, we can confirm that the performance of the inference system can improve even when the system does not monitor Twitter all the time.

Even though the improvement of the accuracy is marginal, the important aspect is that we can reduce the overhead while maintaining good performance. We used Amazon EC2 pricing to calculate the benefits gained by reducing overhead. In Amazon EC2 pricing, \$0.45 is required per hour for the use of general purposes. In Section 5, we used 100 IP addresses for two months to crawl and monitor data. If we use Amazon EC2, $\$0.45 \times 100 \times 60 = \$2,700$ is required to do the same experiment. Therefore, we can save \$540 to do the experiment in Section 5 because, on average, Twitter users do not post tweets for five hours a day. If this remained hours are used for additional monitoring, the advanced inference system can monitor one more user because 25 hours remain. These benefits make the attackers can attack more users at the same time.

7 Discussion

In this section, we describe limitations and countermeasures of our inference system. We also introduce how attackers can exploit our attack to conduct advanced attacks to motivate service providers to develop countermeasures against our attack.

7.1 Limitations

Our inference attack method has some limitations due to the restrictions in the given information. First, we cannot guarantee the correctness of the given location information because some users do not reveal their exact location information on Twitter. Second, the given browser and platform information is also restricted because some client applications do not reveal the exact platforms that they support. Third, even when we are able to identify specific Twitter users, many users have the same information as the identified Twitter users have. Therefore, the results of our inference cannot be 100% guaranteed. However, if we can obtain information about a target user from different channels (e.g., if we are personally acquainted with the target,) we can increase the probability of succeeding with our inference attacks.

7.2 Countermeasures

To cope with our attacks, the publishing policy of public click analytics should be changed. A simple measure of prevention is by delaying the update to the click analytics of shortened URLs. If the click analytics is updated every minute or every tens of minutes, the changes in the click analytics may include multiple click events so that inference attacks have difficulties in differentiating an individual from the group of click events. In addition, providers can add noise information to the click analytics in order to prevent exact inference, as the differential privacy does [9].

7.3 Applications

Our inference attack can be used to develop a number of applications. First, attackers can use it for targeted marketing or spamming because they can infer their target users' preferences by using which URLs they click on, such as music interests, political inclination, or favorite products. Second, attackers can use it for cyberstalking on Twitter. Anyone can stalk a target user by creating a Twitter account that follows everyone whom the target user follows to peek at the tweets that the target user receives. Third, active inference attacks are possible as in Section 1. If an attacker creates a shortened URL and sends the shortened URL to the target user, who then clicks on the shortened URL, the attacker can obtain information, such as the target user's current location and platform, from the click analytics. Consequently, service providers should consider these cases to protect their customers from our attack.

8 Related Work

8.1 Browser History Stealing

There are several types of history stealing attacks. First, attackers exploit Cascading Style Sheet (CSS) visited styles. They use the fact that browsers display visited links differently from unvisited links. This attack is first introduced in 2000 [3], and it has been discovered several times [4], [8]. In the academic community, Janc *et al.* propose a CSS based history stealing attack [16]. They analyze behaviors of each browser related to CSS visited styles and build a system to detect browsing history of users efficiently.

Second, attackers exploit browser and DNS cache to conduct history stealing attacks. Felten *et al.* describe attack methods using browser and DNS cache [10]. They measure the time required to load web pages and to look up DNS. Jackson *et al.* enforce a same-origin policy to prevent history stealing attack [14]. They implement a Firefox extension to enforce the policy on the browser cache and visited link. Jakobsson *et al.* personalize URL to prevent sniffing of browser cache and history [15]. Attackers sniff DNS caches to infer visited sites of users. Grangeia proposes a technique of DNS cache sniffing [11]. Krishnan *et al.* describe how to leak user's privacy by using DNS prefetching [17].

Third, some researchers propose attack methods to steal browsing history using user interactions and side-channels. Weinberg *et al.* exploit CAPTCHA to deceive users and to induce user's interaction [29]. They also use a webcam to detect the light of the screen reflected at the user's face, which can be used to distinguish the colors of visited from those of unvisited links.

The conventional history stealing attacks usually assume that victims visit a malicious web page or victims are infected by malware. However, our inference attacks do not need to make these assumptions. Our inference attacks only use the combinations of publicly available information, so anyone can be an attacker or a victim.

8.2 Privacy Leaks from Public Information

Previous studies have considered attack techniques that cause privacy leaks in social networks, such as inferring private attributes and de-anonymizing users. Most of them combine public information from several different data sets to infer hidden information. Some studies introduce de-anonymizing attacks in social networks. Backstrom *et al.* [1] try to identify edges existence in an anonymized network, and Narayanan and Shmatikov [20] identify Netflix records of known users using only a little bit of data about users. Furthermore, they combine their results with IMDb data and inferred user's political preferences or religious view. Narayanan and Shmatikov [21] also prove that users who have accounts in both Twitter and Flickr can be recognized in the anonymous Twitter graph. Wondracek *et al.* [30] propose a de-anonymized attack using group membership information obtained by browser history stealing attack.

Other studies consider how to infer the private attributes of users in social networks. He *et al.* [12] and Lindamood *et al.* [18] build a Bayesian network to predict undisclosed personal attributes. Zheleva and Getoor [31] show how an attacker can exploit a mixture of private and public data to predict private attributes of a target user. Similarly, Mnislove *et al.* [19] infer the attributes of a target user by using a combination of attributes of the user's friends and other users who are loosely (not directly) connected to the target user. Calandrino *et al.* [5] propose algorithms inferring customer's transactions in the recommender systems, such as Amazon and Hunch. They combine public data of the recommender systems and some of the transactions of a target user in order to infer the target user's unknown transactions. Chaabane *et al.* [6] propose an inference attack to predict undisclosed attributes by using only music interests. They derive semantics using Wikipedia ontology and measured the similarity between users.

9 Conclusion

In this paper, we proposed inference attacks to infer which shortened URLs clicked on by a target user. All the information needed in our attacks is public information: the click analytics of URL shortening services and Twitter metadata. To evaluate our attacks, we crawled and monitored the click analytics of URL shortening services and Twitter data. Throughout the experiments, we have shown that our attacks can infer the candidates in most cases.

10 Acknowledgements

A preliminary version of this paper was presented at WWW 2013 [23]. This research was supported by the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2013-H0301-13-3002)

References

[1] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. In *Proc. 16th Int'l World Wide Web Conf. (WWW)*, 2007.

[2] D. boyd, S. Golder, and G. Lotan. Tweet, tweet, retweet: Conversational aspects of retweeting on twitter. In *Proc. 43rd Hawaii International Conference on System Sciences (HICSS)*, 2010.

[3] Bugzilla. Bug 57351: css on a:visited can load an image and/or reveal if visitor been to a site, 2000. `https://bugzilla.mozilla.org/show_bug.cgi?id=57351`.

[4] Bugzilla. Bug 147777: visited support allows queries into global history, 2002. `https://bugzilla.mozilla.org/show_bug.cgi?id=147777`.

[5] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. “you might also like:” privacy risks of collaborative filtering. In *Proc. IEEE Symp. Security and Privacy (S&P)*, 2011.

[6] A. Chaabane, G. Acs, and M. A. Kaafar. You are what you like! information leakage through users’ interests. In *Proc. 19th Network and Distributed System Security Symp. (NDSS)*, 2012.

[7] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: A content-based approach to geo-locating twitter users. In *Proc. 19th ACM International Conference on Information and Knowledge Management (CIKM)*, 2010.

[8] A. Clover. Css visited pages disclosure, 2002. `http://seclists.org/bugtraq/2002/Feb/271`.

[9] C. Dwork. Differential privacy. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[10] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *Proc. 7th ACM Conf. Computer and Comm. Security (CCS)*, 2000.

[11] L. Grangeia. Dns cache snooping or snooping the cache for fun and profit. In *SideStep Seguranca Digitial, Technical Report*, 2004.

[12] J. He, W. W. Chu, and Z. V. Liu. Inferring privacy information from social networks. In *Proc.4th IEEE international conference on Intelligence and Security Informatics (ISI)*, 2006.

[13] B. Hecht, L. Hong, B. Suh, and E. H. Chi. Tweets from justin bieber’s heart: The dynamics of the location field in user profiles. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2011.

[14] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *Proc. 15th Int’l World Wide Web Conf. (WWW)*, 2006.

[15] M. Jakobsson and S. Stamm. Invasive browser sniffing and countermeasures. In *Proc. 15th Int’l World Wide Web Conf. (WWW)*, 2006.

[16] A. Janc and L. Olejnik. Web browser history detection as a real-world privacy threat. In *Proc. 15th European conference on Research in computer security (ESORICS)*, 2010.

[17] S. Krishnan and F. Monrose. Dns prefetching and its privacy implications: When good things go bad. In *Proc. 3rd USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, 2010.

[18] J. Lindamood, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. Inferring private information using social network data. In *Proc. 18th Int’l World Wide Web Conf. (WWW)*, 2009.

[19] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks. In *Proc. 3rd ACM International Conference on Web Search and Data Mining (WSDM)*, 2010.

[20] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse dataset. In *Proc. IEEE Symp. Security and Privacy (S&P)*, 2008.

[21] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Proc. IEEE Symp. Security and Privacy (S&P)*, 2009.

[22] Semiocast. Twitter reaches half a billion accounts more than 140 millions in the u.s., 2012. `http://semiocast.com/publications/2012_07_30_Twitter_reaches_half_a_billion_accounts_140m_in_the_US`.

[23] J. Song, S. Lee, and J. Kim. I know the shortened urls you clicked on twitter: Inference attack using public click analytics and twitter metadata. In *Proc. 22nd Int’l World Wide Web Conf. (WWW)*, 2013.

[24] Twitter blog. Links and twitter: Length should’t matter, 2010. `http://blog.twitter.com/2010/06/links-and-twitter-length-shouldnt.html`.

[25] Twitter blog. One million registered twitter apps, 2011. `http://blog.twitter.com/2011/07/one-million-registered-twitter-apps.html`.

[26] Twitter blog. Shutting down spammers, 2012. `http://blog.twitter.com/2012/04/shutting-down-spammers.html`.

[27] Twitter developers. t.co redirection behavior, 2012. `https://dev.twitter.com/docs/tco-redirection-behavior`.

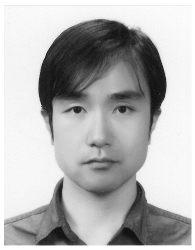
[28] Twitter developers. The t.co url wrapper, 2012. `https://dev.twitter.com/docs/tco-url-wrapper`.

[29] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proc. IEEE Symp. Security and Privacy (S&P)*, 2011.

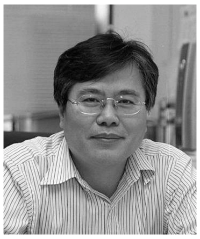
[30] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *Proc. IEEE Symp. Security and Privacy (S&P)*, 2010.

[31] E. Zheleva and L. Getoor. To join or not to join: The illusion of privacy in social networks with mixed public and private user profiles. In *Proc. 18th Int’l World Wide Web Conf. (WWW)*, 2009.

Jonghyuk Song received the B.S. in computer science and engineering from Pohang University of Science and Technology (POSTECH), Korea in 2008. He is a Ph.D. student in computer science and engineering at POSTECH. His research interests include Web and social network service security.



Sangho Lee received the B.S. in computer engineering from Hongik University, Korea, in 2006, the M.S. degree in computer science and engineering from Pohang University of Science and Technology (POSTECH), Korea, in 2008, and the Ph.D. degree in computer science and enginnering from POSTECH in 2013. He is currently a post-doctoral research associate in the Department of Computer Science and Engineering, POSTECH. His research interests include Web and online social network security, system security, and privacy protection.



Jong Kim received the B.S. in electronic engineering from Hanyang University, Korea, in 1981, the M.S. degree in computer science from the Korean Advanced Institute of Science and Technology, Korea, in 1983, and the Ph.D. degree in computer engineering from Pennsylvania State University in 1991. He is currently a professor in the Division of IT Convergence Engineering, Pohang University of Science and Technology (POSTECH), Korea. From 1991 to 1992, he was a research fellow in the Real-Time Computing Laboratory of the Department of Electrical Engineering and Computer Science, University of Michigan. His major areas of interest are fault-tolerant computing, parallel and distributed computing, and computer security.

Appendix A
Problem Formalization

A.1 Attack Model

In this section, we formalize the inference attack described in the previous sections. The inference attack consists of three components: service, target user, and adversary.

- **Service:** Each service releases the metadata and/or usage statistics of their users. The service updates the data in real time and releases it in an aggregate form to prevent privacy leakage. Services can be connected to each other. The *connected services* consists of a *main service* used primarily (e.g., online social networks and blogs) and *third party services* supporting the usages of the main service (e.g., URL shortening services, location services, and photo sharing services).
- **Target user:** Any user who simultaneously uses the *connected services* can be a target user. A target user should have public information that can be exploited to infer his private usage.
- **Adversary:** In our inference attack, an adversary does not need to have private information or to exploit complicate techniques. Anyone who can access the released information can be an adversary. An adversary constantly monitors the information released by the services. Then, he tries to extract the private usage information of a target user by exploiting the information released by the connected services.

The above three components are essential to our inference attack. Another essential requirement for the inference attack is that there should be *overlapping information* between the information released by the connected services. If the overlapping information corresponds with the information of a target user, an adversary can know that the target user has used the connected services.

We define our *inference attack* based on the components as follows:

Definition 1. ***Inference attack*** *tries to detect a user who simultaneously uses the connected services by matching the overlapping information with the user information.*

The outcome of an inference attack is a set of candidate users whose information corresponds with the overlapping information. The accuracy of our inference attack is in inverse proportional to the number of candidate users because only one user is a target user among the set of candidate users. Therefore, the accuracy of an inference attack is $\frac{1}{|candidate\ users|}$. The lower bound of the accuracy is $\frac{1}{N}$ when the number of the candidate users is maximum which means all users are candidate users. Therefore, the inference system try to reduce the number of candidate users as much as possible for better performance.

A.2 Algorithms

In this section, we propose algorithms to apply our inference attack in general situations. We first define user and data models. Let U be user information released by the main service. Let D be a dataset released by the third party services. To protect the user's privacy, third party services provide the *online* dataset D in aggregate form which consist of attributes a, values v and count of them c. Let A_U be an attribute set of U and A_D be an attribute set of D. We define U, D and their attribute sets as follows:

$$\begin{aligned} A_U &= \{a \mid a \text{ is an attribute of } U\} \\ A_D &= \{a \mid a \text{ is an attribute of } D\} \\ U &= \{(a : v) \mid a \in A_U,\ v \text{ is a value of } a\} \\ D &= \{(a : v,\ c(t)) \mid a \in A_D,\ v \text{ is a value of } a, \\ &\qquad c \text{ is the counter of a tuple } (a : v) \text{ at time } t\} \end{aligned}$$

Algorithm 1 Inference attack for a target user

```
Input: $A_C = A_U \cap A_D$
$u = \{(a : v) \mid a \in A_C,\ v$ is a value of $a\}$ and $u \subseteq U$
$d(t) = \{(a : v,\ c(t) \mid a \in A_C,\ v$ is a value of $a$, $c(t)$ is a counter of $(a : v)$ tuple at time $t\}$ and $\exists(a : v,\ c(t)) \in D$
Output: Inferred time the user has used the service

$history = \{\}$
foreach $observation\ time\ at\ t$ do
    $\Delta d(t) = \{(a : v) \mid \exists(a : v,\ c(t)) \in d(t)$ s.t $(c(t) - c(t-1)) \geq 1\}$
    if $u \subseteq \Delta d(t)$ then
        $history = history \cup \{t : u\}$
    end
end
return $history$
```

Algorithm 1 shows the procedure of the inference attack in the case of a single target user and a single third party service. Usually, the attributes of U and D differ from each other. Therefore, the system has to calculate a set of common attributes of A_U and A_D, which is defined as A_C. $\Delta d(t)$ is the differences between $d(t)$ and $d(t-1)$. If a user u is a subset of the $\Delta d(t)$, the inference system infers that the user has used the service at time t. In this way, we can obtain the service usage history of the user, which is defined as follows:

$$history = \{(t : u) \mid u \text{ exist at time } t\},$$

where $(t : u)$ means that a user u used the service at time t.

Algorithm 2 shows the procedure of the inference attack on multiple users. When our inference system obtains $\Delta d(t)$, the system compares it with each user. If u_i is a subset of $\Delta d(t)$, the system adds u_i into inferred history with time t. Finally, we can have the usage history that shows which users used the service at time t.

Algorithm 3 shows the procedure of the inference attack in case of multiple third parties when each dataset of third party services has different attributes and has few attributes than the user has. In some cases, it is not enough to infer user's behavior using only one third party database due to a lack of information. Algorithm 3 uses multiple dataset to

Algorithm 2 Inference attack for multiple target users

```
Input: $A_C = A_U \cap A_D$
$u_1, u_2, ..., u_n$ : $n$ users
$u_i = \{(a : v) \mid a \in A_C$, $v$ is a value of $a\}$ and $u_i \subseteq U$
$d(t) = \{(a : v,\ c(t)) \mid a \in A_C$, $v$ is a value of $a$, $c(t)$ is the counter of a tuple $(a : v)$ at time $t\}$ and $\exists(a : v,\ c(t)) \in D$
Output: Inferred time and users

$history = \{\}$
foreach observation time at $t$ do
    $\Delta d(t) = \{(a : v) \mid \exists(a : v,\ c(t)) \in d(t)$ s.t $(c(t) - c(t-1)) \geq 1\}$
    for $i = 1$ to $n$ do do
        if $u_i \subseteq \Delta d(t)$ then
            $history = history \cup \{t : u_i\}$
        end
    end
end
return $history$
```

improve the accuracy of the inference attack for such cases. We define U, D_i and attribute sets as follows:

Algorithm 3 Inference attack with multiple third party services

```
Input: $A_{C_i} = A_U \cap A_{D_i}$
$u = \{(a : v) \mid a \in A_U$, $v$ is a value of $a\}$ and $u \subseteq U$
$d_1, d_2, ..., d_n$ : $n$ datasets of the third party services
$d_i(t) = \{(a : v,\ c_i(t) \mid a \in A_{C_i}$, $v$ is a value of $a$, $c_i(t)$ is a counter of $(a : v)$ tuple at time $t\}$ and $\exists(a : v,\ c_i(t)) \in D_i$
Output: Inferred time the user has used the services

$history = \{\}$
foreach observation time at $t$ do
    $\forall\ i : d_i(t) = \{(a : v) \mid \exists(a : v,\ c_i(t))$ s.t $(c_i(t) - c_i(t-1)) \geq 1\}$
    if $(\forall\ i : \Delta d_i(t) \subseteq u)$ then
        $history = history \cup \{t : u\}$
    end
end
return
```

$$
\begin{aligned}
A_U &= \{a \mid a \text{ is an attribute of } U\} \\
A_{D_i} &= \{a \mid a \text{ is an attribute of } D_i\} \\
U &= \{(a : v) \ \mid \ a \in A_U,\ v \text{ is a value of } a\} \\
D_i &= \{(a : v,\ c(t)_i) \ \mid \ a \in A_{D_i},\ v \text{ is a value of } a, \\
&\qquad c_i \text{ is a counter of } (a : v) \text{ tuple at time } t\}
\end{aligned}
$$

If the information of a user is recorded in all dataset of the third party services at time t, the inference system can know that the user used all services at time t.