# Online Social Honeynets: Trapping Web Crawlers in OSN

Jordi Herrera-Joancomartí and Cristina Pérez-Solà

Dept. d'Enginyeria de la Informació i les Comunicacions
Escola d'Enginyeria
Universitat Autònoma de Barcelona
08193 Bellaterra, Catalonia, Spain
{jherrera,cperez}deic.uab.cat

**Abstract.** Web crawlers are complex applications that explore the Web with different purposes. Web crawlers can be configured to crawl online social networks (OSN) to obtain relevant data about its global structure. Before a web crawler can be launched to explore the web, a large amount of settings have to be configured. This settings define the behavior of the crawler and have a big impact on the collected data. The amount of collected data and the quality of the information that it contains are affected by the crawler settings and, therefore, by properly configuring this web crawler settings we can target specific goals to achieve with our crawl. In this paper, we analyze how different scheduler algorithms affect to the collected data in terms of users' privacy. Furthermore, we introduce the concept of online social honeynet (OShN) to protect OSN from web crawlers and we provide an OShN proof-of-concept that achieve good results for protecting OSN from a specific web crawler.

**Keywords:** privacy, social networks, web crawling, graph mining, social honeynets.

## 1 Introduction

The increasingly popularity of online social networks (OSN) has lead them to become an important part of people's everyday communication. With millions of individuals who use OSN to share all kinds of contents, privacy concerns of how all this content is managed have arisen. Content shared in an OSN varies from trivial text messages to compromising photographies but, in either of those cases, users expect to control their shared data with their profile's visibility configuration. In addition to this personal data, users in OSN create relationships which can also be considered sensitive data from themselves. Moreover, the discovering of these relationships can also produce other data revelation, what makes link privacy an important issue to preserve in social networks.

OSN information can be obtained by crawling the profiles of users in the network. Web crawlers are complex applications that explore the Web with different purposes and they can be configured to crawl OSN to obtain both user and link information. When crawling online social networks, many choices have to be made in order to set up the crawler that is going to be used to obtain all the information from a social networking site. This configuration choices conform the crawler settings and, as we are going to see, they are the key to accomplish the desired crawling goal. Specifically, the choice of the next-node-to-crawl (determined by the scheduler algorithm) is a critical point, since it will determine largely which part of the network is going to be obtained and, therefore, which level of exposure will suffer the online social network users.

The contribution of this paper is twofold. On one hand, we detail the privacy implications that imply the election of different scheduler algorithms for web crawlers. On the other hand, we introduce the concept of Online Social Honeynet (OShN) to provide some level of protection against attacks performed by web crawlers. We provide a proof-of-concept of the feasibility to design appropriated OShN that can prevent specific web crawler configurations.

The rest of the paper is organized as follows. First, we present the state of the art and we describe the basic architecture of a web crawler with special emphasis in the scheduler module, detailing some of the scheduler algorithms that a crawler may implement. Later on, we discus the privacy threats that each of these scheduler algorithms suppose for the online social network users. After that, the concept of Online Social Honeynet is introduced in order to mitigate these privacy risks originated from the usage of web crawlers. Finally, we present the conclusions and provide some guidelines for further research.

## 2 State of the Art

The Web crawling problem has been widely studied in the scientific literature and in the practical arena. Architectures for web crawlers are proposed in [1] and [2]. These studies are centered on obtaining a fully scalable web crawler which can be used to crawl the entire Web. Detailed analysis on the bottlenecks of the crawling architectures can also be found in previous articles. Architectures for distributed web crawlers have been proposed too ([3] and [4]).

Web crawling scheduler algorithms have also been studied in depth, mostly for its use on Internet search engines. However, less attention has been paid, until now, to the specific scenario of crawling online social networks. At present time, studies in OSN web crawling deal with different related problems from algorithm performance to quality of collected data.

In [5], authors evaluate how different parameters of the crawler algorithms affect crawling efficiency (defined previously in [6]) as well as the quality of collected data. Biases produced by certain schedulers can be avoided by selecting the proper scheduler algorithm as is shown in [7], where a random sample of Facebook users is collected using a Metropolis-Hasting Random Walk (MHRW). They also demonstrate that metrics obtained with MHRW largely differ from those obtained with BFS, which remarks the importance of properly selecting the crawler scheduler algorithm based on the crawling purpose. Graphs retrieved by different collection techniques are also compared for Twitter network in [8].

Large-scale measurement of online social networks has been done in [9], where four of the most popular social networks were crawled in depth. Analyzing the obtained data the authors are able to confirm that online social networks satisfy the power-law, small-world and scale-free properties.

Although some of this crawling literature refers to users' privacy, comparisons made by different crawling algorithms are centered in their effect on classic graph metrics or on crawling efficiency, but, at our best knowledge, no work about how crawling algorithms affect user's privacy can be found in this crawling literature. On the other hand, a similar problem appears in privacy literature, where user's privacy is analyzed in detail but no references to crawler algorithms can be found.

Privacy implications of social networks have been a popular topic in recent years. Link privacy has been studied in [10] and [6]. In [10], Backstrom *et al* present several attacks on edge privacy. These attacks allow an adversary to reidentify a set of targeted users from a single anonymized copy of the network. In [6], edge privacy is studied from the point of view of the number of compromised accounts needed to expose as much nodes as possible depending on the lookahead of the network. Lookahead is defined as the distance from which a user can see his friends links.

Theoretical work centered on maintaining privacy when releasing network data sets has also been done. In [11], the authors quantify the privacy risks associated with different network release scenarios and propose an anonymization technique that leads to substantial reduction of the privacy threat. In [12], authors propose several strategies for preventing link re-identification in anonymized graphs. In [13], authors assume that an adversary knows the neighborhood of some target individuals and present an anonymization algorithm. In [14], other anonymization techniques are proposed, now considering that the adversary knows the degree of certain nodes a priori. Much effort has also been done in reidentification algorithms for anonymized social graphs in [15], where the authors present a deanonymization algorithm based on the usage of publicly available auxiliary information.

## 3 Web Crawling Architecture

Web crawlers are programs that automatically explore web pages in a methodical manner. Web crawlers start the search in one or more URLs, which are called seeds, and explore them in order to find new URLs to search for, until they reach a predefined termination condition. When used to crawl OSN, web crawlers start from an initial user, or list of users, and discover other users of the network by following their social relationships.

Although the architecture of a web crawler is not a fixed one and different solutions have been proposed to optimize the crawling process, the basic architecture of a web crawler can be explained by defining its five essential modules:

1. The **downloader** is the interface between the Web (or, in our case, the OSN that is being explored) and our crawler. Its job is to download a web page and pass it to the parser.

2. The **parser** is in charge of analyzing the page that has been downloaded and extract useful information and links to other pages.
3. The **storage device** keeps record of the crawled information, information about a user that can be found in his profile (*e.g.* name, location or birth date) together with links to other pages that are, in fact, links to other users' profiles which define user relationships inside the OSN.
4. The **queue** contains all the links to other users' profiles found when crawling every user that are awaiting to be explored.
5. The **scheduler** is responsible for selecting which user, from the ones in the queue, is going to be explored and communicating its decision to the downloader, completing the crawling cycle.

Based on the described crawling process of the OSN, users can be classified in three different categories: crawled, discovered or hidden. A **crawled user** is the one that all his profile's information and all his friends are known to the crawler (we denote by $V_{crawl}$ the subset of crawled users). A **discovered user** is the one that his presence and at least one relationship is noticed by the crawler but is not a crawled user ($V_{disc}$ denotes the subset of discovered users). Finally, a **hidden user** is the one that the crawler is not even aware of his existence ($V_{hidd}$ denoting this subset). We also use $n_* = |V_*|$ to describe de cardinality of each set.

## 3.1 Scheduler Algorithms

The scheduler algorithm is the most critical part of a web crawler since its definition and configuration impacts in important aspects of a web crawler, like performance, efficiency or collected data. In this section, we describe different scheduler algorithms. The goal of this section is to provide a comprehensive description of the most frequently used scheduler algorithms in order to discuss, in next section, their implications on the collected data that, in fact, determines the users privacy. For that reason, no detailed measures on performance or efficiency are included. Interested readers can review [6] or [5] for an exhaustive study on these characteristics for different scheduler algorithms.

– **Breath-First Search (BFS)** algorithm acts as a simple queue, where the first nodes to be crawled are the first that have been discovered. Newly discovered nodes are appended to the end of the queue, thus previously discovered nodes are crawled sooner than the new ones.
– **Depth-First Search (DFS)** algorithm works as a traditional stack, where the first nodes to be crawled are the last ones that have been discovered. Newly discovered nodes are added at the top of the stack, thus they are going to be explored sooner than previously crawled nodes.
– **Greedy** algorithm selects as the next node to be crawled the one with the highest degree from all $V_{disc}$ nodes. Depending on how this degree is computed, we can distinguish three different greedy algorithms:
  - **Real-degree greedy** takes its decisions based on the real degree of the nodes in the OSN. Notice that using the architecture described above,

  information on the real degree of a node is unknown for discovered nodes so additional requests may have to be done to the OSN in order to use this scheduler. This real-degree greedy definition corresponds to the *hypothetical greedy* algorithm in [5] and would be called *highest-degree-crawler* in the [6] terms.
  - **Explored-degree greedy** uses the actual known degree of the node in the explored subgraph $G_{crawl}$ as the measure to select the next node to crawl. This definition of explored-degree greedy is the same that can be found in [5] under the mere *greedy* name.
  - **Unseen-degree greedy** uses the unseen degree of a node, that is the real degree minus the explored one. Unseen degree corresponds to the number of friends of a node that the crawler is not aware of. This definition of unseen-degree is exactly the same of the *degree-greedy-crawler* used in [6].
- **Lottery** algorithm selects the next node to be crawled with a proportional probability with its degree. This gives more chance to high degree nodes to be selected while maintaining the possibility to select low degree ones. Lottery algorithm can be configured to use any of the previous degrees (real, explored or unseen) in order to make its decisions.

## 4 Privacy Threats Related to Crawling Activity

By crawling an OSN the corresponding social graph can be obtained. Such social graphs may provide very important information about the network and their users, since using appropriated graph mining techniques allows to discover important user characteristics.

When dealing with social graphs, two kinds of user's information can be extracted: node information and edge information. All data about a specific user is considered as node information. Node information includes all details provided in the user's profile on a specific OSN. Such data generally contains information like user name, age, nationality, current location, phone number, marital status, personal web site url and a thumbnail. Moreover, specific content OSN include other information in their users profile like photographies, music or books preferences.

The other kind of user information that can be obtained from the social graph is edge information. The mere existence of edges already offers information about users that are linked through them but, in some networks, this edges can be labeled, thus providing a more in depth information about the relations that they represent. A part from providing information of the relationships between different users, edges can also directly disclose node attributes. For instance, an edge representing a sentimental relationship between two individuals of the same sex would be revealing their sexual orientation.

Although both node attributes and edges may be considered sensitive information that the user wants to control, in this paper we focus on edge privacy since edges suppose an added risk to user's privacy in many different ways. In contrast to node attributes, which disclosure can be configured by the user, protecting edge information involve more than one user and, for that reason, it makes more difficult for the participating users to maintain control on the visibility of this relations [16]. On the other hand, relations between users can be used to detect communities. Communities are groups of nodes which are highly tight together within the network. Detecting and identifying these communities is a usual procedure in social network analysis, since communities facilitate the understanding of network data. Knowing to which communities does a user belong is an excellent way to gain information about the user: family, friends, college or work mates are subgroups that arise from a social network and can be detected from the graph structure itself. Since they do not need the explicit intervention of the user to be created, they entail a new risk for OSN users privacy. Moreover, it has been shown that users belonging to the same clique share common interests, believes or even food habits [17], which are in fact, node attributes. For this reason, node attributes can be induced from information known about other users in the same clique.

Furthermore, edge information has been proved to serve as auxiliary information for many deanonymization attacks ([10], [15]), which makes edges and its attributes an important information to care about. Relations that a user has with others describe that user in a quasi-unique form. Even when all labels have been removed from the graph, its structure is leaking information that can be used to reidentify the nodes. For instance, if an adversary knows how many friends does the victim have and which are the relations among them, the attacker may be able to find this subgraph inside an anonymized release of the whole graph and learn information about the victim and his friends.

## 4.1 Scheduler Implications on Privacy

It seems clear that the corresponding social graph of an OSN is a powerful tool to derive private information of the users. However, due to the actual size of OSN sites, crawling them entirely to obtain the corresponding social graph may not be an affordable option. Having to conform with the obtainment of a partial view, the concept of quality of the collected data of the crawler comes into play. The scheduler algorithm, together with the initial seed of the crawler, is the module of the crawler that determines the path to follow during the crawling process and then the exact data that will be finally retrieved from the OSN.

The quality of collected data is a difficult term to deal with since such quality depends on the objective of the crawling process.

In order to make a comprehensive analysis, we fixed three different and somehow opposite objectives for the crawler (from the attacker's point of view):

– **Objective A:** to determine all links and communities where a specific victim belongs to.
– **Objective B:** to discover general characteristics of the OSN, focused on identifying communities.
– **Objective C:** to discover the maximum number of nodes of the network.

Notice that while objective A is centered on attacking a single user, objectives B and C target the whole network but with different purposes in mind.

For each scheduler algorithm, we analyze the achievement of these objectives in terms of cohesive subgroups identification (A and B) or crawling efficiency (C). For cohesive subgroups identification, we focus on finding cliques and $k$-plexes [17] since this structure relaxes the strong familiarity conditions expressed in a clique but, at the same time, still provide the properties of reachability and robustness in the resulting cohesive group. For crawling efficiency, we will use the metric defined previously in [6], where efficiency is defined as the number of discovered nodes divided by the number of crawled nodes.

**Breath-First Search.** Using a BFS algorithm with only one initial user as seed allows the crawler to explore the $k$-neighborhood of the seed, that is, to crawl all nodes at distance $k$ (starting at $k = 1$) from the seed and, therefore, discover all nodes at distance $k+1$. Then, the collected data obtained using a BFS scheduler algorithm is of high quality regarding *Objective A*, since an accurate view of the OSN centered on the victims will be obtained.

However, BFS performs poorly with respect to *Objective B*. The sequentiality of the BFS with respect to the neighbor distance $k$ does not allow to move the crawler to specific nodes belonging to interesting communities, and then the collected data cannot be taken as a representative of the OSN since it is focused on a particular part.

In BFS algorithm, no special attention is paid to higher degree nodes thus BFS does not offer advantages regarding *Objective C*.

**Depth-First Search.** As DFS scheduler algorithm tries to get as far as possible from the initial seed, neither the neighborhood of the seed nor subgroup structures will not be formed easily when the value $n_{crawl}$ is low with respect $n_V$. In fact, cliques that are actually found by this crawling method will be small, usually with just 3 nodes. For that reason, collected data of a crawler with DFS scheduler algorithm does not provide quality information regarding neither *Objective A* nor *Objective B*.

DFS does not take into account node degrees neither, but crawling efficiency is slightly better for DFS than for BFS. The reason is that, as the crawler tries to get far away from the seed, crawled nodes tend to have a few friends in common, thus for the same $n_{crawl}$ more $n_{disc}$ are obtained. Thus DFS performs better than BFS with respect to *Objective C*.

**Real-degree greedy.** Real-degree greedy moves quickly through the largest degree node, and once reached, the algorithm provides large numbers of cliques and $k$-plexes since at each iteration a maximum number of edges are added to the crawled graph. For this reason, this algorithm provides a good data quality regarding *Objective B*. However, real-degree greedy is not suitable to reach *Objective A*, unless the victim is the highest degree node. In fact, higher degree nodes are very vulnerable against this scheduler algorithm since they are reached with few iterations independently of the used seed.

As first nodes selected to be crawled are the ones with higher degrees, graphs obtained with real-degree greedy always present a high mean degree, which is much more bigger than the real mean degree of the complete OSN. Selecting this high degree nodes leads to obtain high efficiency, thus this algorithm is adequate to reach *Objective C*.

**Explored-degree greedy.** In the explored-degree greedy, first nodes to be crawled are the ones that are more connected to already crawled ones. In contrast to the real-degree greedy, explored greedy also move towards the highest degree node but more slowly, finding the cliques and $k$-plexes that are in the path between the initial seed and the highest degree node. With these properties, explored-degree greedy algorithm is suitable to achieve *Objective B* although the speed at which cliques and $k$-plexes are discovered is much more lower that with real-degree greedy. Regarding *Objective A*, the explored-degree greedy does not provide a good strategy since it does not guarantee that the crawl is centered on the seed and then, the initial seed may not belong to the cohesive subgroups that are retrieved. However, in comparison with real-degree greedy, explored-degree greedy keeps the crawler closer to the seed and then, in terms of *Objective A*, explored-greedy performs better than real-greedy.

**Unseen-degree greedy.** The first users to be crawled with unseen-degree are the ones that have a high real degree and a small explored degree. In the first iterations of the crawler, unseen-degree and real-degree perform similar, moving quickly towards the highest degree node. At later stages of the crawler, the unseen-degree greedy achieves better efficiency since it discovers more new nodes than the real-degree. However, since the discovered nodes do not provide much information into the crawled graph until they are crawled, the numbers of cliques and $k$-plexes, and its sizes are equivalent to the ones obtained with real-degree. For that reason, performance of unseen-greedy with respect to *Objectives A* and *B* is equivalent to real-degree greedy.

Selecting the highest unseen degree node as the first node to crawl results in selecting the node that would lead the crawler to discover the maximum amount of new nodes when it is crawled. Then, unsee-degree greedy performs better than the above algorithms regarding *Objective C*.

**Lottery.** The random effect introduced in the lottery schedulers gives a chance to select low degree nodes as the next-node-to-crawl. As a consequence, for the same number of $V_{crawl}$ nodes, lottery will discover more nodes than BFS, random list or DFS but less than greedy schedulers. So we can affirm that lottery performs better than BFS, random list and DFS regarding *Objective C* but worse than greedy. The same happens with found cliques and $k$-plexes when using the explored degree as a selection measure. In this case, lottery will find more cliques than DFS or random list but less than greedy algorithms. Much like the explored-degree greedy case, explored-degree lottery also presents the problem that the initial seed may not belong to the found cliques, which can suppose a problem when the pursued goal is *Objective A*.

Like in the greedy case, lottery tends to select as next node to crawl the ones with highest degrees (whatever the chosen degree is used), resulting on a higher mean degree in $G_{crawl}$ than the actual graph $G$ mean degree. However, this effect is less pronounced in the lottery case because its random component that gives a chance to low degree nodes to be selected. As a consequence, lottery performs worse than greedy algorithms regarding *Objective B*.

## 5 Online Social Honeynets

As we have seen, web crawling supposes a big risk for users privacy. OSN contain enormous amounts of personal data which is, in most cases, publicly available to anyone who is interested in it. Web crawlers can be used as a tool to collect all this data. For this reason, it is important to be able to defend an OSN from automated web crawlers which try to obtain information about its users.

The first trivial approach to avoid these risks is to deny the access to the network for web crawlers. In order to do so, it is needed to distinguish between web crawlers and other kinds of accesses (usually web browser requests) to the network. Although some web crawlers identify themselves as so via the User Agent field in the HTML protocol, it is easy to forge the requests in order to simulate that they are made by a common browser. Consequently, we can not rely on the HTML User Agent to tell the difference between web crawlers and non web crawlers.

It is also possible to try to forbid the access to web crawlers by banning the public access to the network. However, this is a difficult task to perform without affecting the usability of the network. It is possible to configure the network in such manner that only registered users are allowed to obtain information about other users. In addition, the information that a user can obtain of another user can be constrained depending on the distance between this users. For instance, a sample configuration may be to allow a user to obtain all the information that the network has of a direct friend, only the degree of a user which is a friend of a friend and none information at all about the rest of the users of the network.

However, even when the network is closed and the neighbors of a targeted user can only be obtained by users in the network at a fixed distance $l$ of this targeted user, published studies [6] show different strategies to maximize the portion of the network discovered depending on the value of the lookahead $l$. All the presented attacks require that the attacker subverts some user accounts to obtain information of its friends. The authors show that for lookahead values higher than 2, the number of subverted accounts needed to discover the 80% of the nodes of the network is less than 100.

Furthermore, there are some OSN whose own properties or objectives make them impossible to be build under a closed paradigm network. This is the case, for example, of Twitter, whose slogan describes it as “the best way to discover what’s happening on your world”. How could be this accomplished by limiting the disclosure of all comments to just the users friends?

Another different approach to try to forbid the access for web crawlers is to try to limit the number of accesses to the network done by the same IP address.

Although this may seem a good strategy, it can be easily circumvented by using anonymizing techniques that mask the source IP address.

As we have seen, neither making the OSN a closed network nor limiting the number of accesses that can be done by the same IP address per unit of time are feasible solutions to our problem. For this reason, some other techniques have to be designed to limit the information that crawler may obtain from OSN. In traditional web crawling literature, web crawler traps are known to cause troubles to web crawlers [1]. Crawler traps are URLs that cause the crawler to crawl indefinitely. In the traditional Web, some crawler traps can be created unintentionally. For example, symbolic links within a file system can create cycles. Other crawler traps are produced intentionally. For instance, CGI programs that dynamically generate an infinite Web of documents. We propose a similar approach to protect OSN from web crawlers by introducing the idea of Online Social Honeynets.

Online Social Honeynets (OShN) are, much like traditional honeynets, a set of users in the network whose objective is to attract and defend the network from attackers that want to retrieve information from the network. Also like traditional honeynets, OShN consist of a set of users that appear to be part of the network with information of value to the attackers but they are actually isolated and monitored. OShN also extents the concept of Social Honeypot introduced in [18] where fake users are created in OSN to detect spam profiles and distinguish social spammers from legitimate users.

So given a social graph $G = (V, E)$ that represents an entire OSN, an OShN can be modeled as a social graph $G_h$, which consist on a fake set of users $V_h$, its relationships $E_h$, and a set of honeynet bridges $E_b$ that will link the real graph $G$ with our honeynet graph $G_h$ (see Figure 1). Then, the disclosed network can be modeled as a social graph $G_d = (V_d, E_d)$ containing all nodes $V_d = V \cup V_h$ from both graphs and all edges $E_d = E \cup E_h \cup E_b$ from both graphs plus the honeynet bridges. Notice that we keep the edges defining the honeynet bridges outside $G$ and $G_h$, since, as we describe later, such bridges play an important role for the objective of the OShN. Nodes in $G_h$ incident to some edge in $E_b$ are called exterior nodes while nodes in $G_h$ without any connection to $G$ will be called interior nodes.

Although it is obvious that the idea of OShN can be used for different purposes, our main goal is to design an OShN that may provide some protection from web crawlers, minimizing the useful information that the web crawler may obtain from the OSN.

In order to protect OSN from web crawlers, the OShN should be able, first of all, to attract web crawlers and, later on, to keep the crawler in the boundaries of the OShN, $G_h$. Notice that with this approach, OSN providers do not have to be concerned anymore about blocking the access to web crawlers since they will be attracted and trapped by the honeynet and, therefore, will not be able to obtain information of the real users of the network.

Let $t_a$ be the time that our OShN needs to attract the crawler, that is the time needed to reach one of the honeynet nodes in $V_h$. Let $t_t$ be the time our

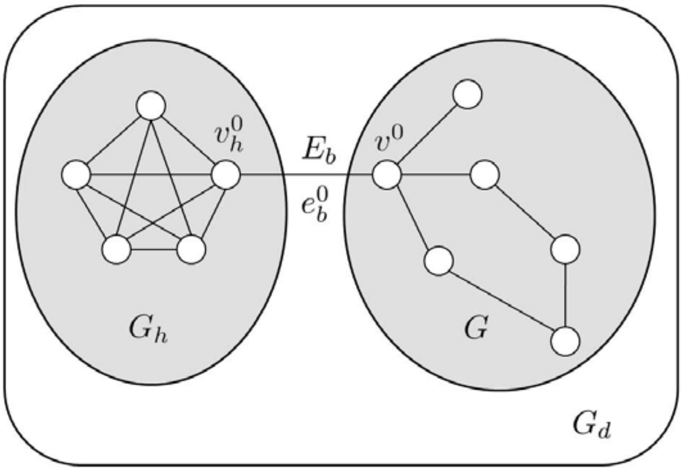

**Fig. 1.** Online Social Honeynet

OShN could trap the crawler. Then, $t_a$ and $t_t$ determine the amount of correct information the crawler may obtain from the OSN and our OShN design should focus on by minimizing $t_a$ and maximizing $t_t$, since in this way we could achieve a good level of protection. The design of such a OShN that achieve such objectives is not an easy task an, obviously, it is not likely that a single OShN could provide such protection for all different web crawler configurations. In fact, as we discuss in the next subsection, the design of an effective OShN is related with the exact scheduler algorithm that the web crawler uses to crawl the OSN.

### 5.1 An Online Social Honeynet to Protect OSN from Greedy Schedulers

In this section, we present a proof-of-concept of an OShN in order to show the feasibility of the idea. We focus our OShN to be resistant against attacks of a web crawler configured using a real-degree greedy which represent a threat for OSN since it achieves a high efficiency rate as it has been proven in [5]. Furthermore, this algorithm is suitable to obtain a general view of the OSN, as it has been pointed out in the previous section 4 and provides an important number of cliques and $k$-plexes.

In order to define our OShN, we make the following assumptions. Firstly, our OShN is static in the sense that elements in $V_h$ , $E_h$, and $E_b$ remain unchanged during the crawling. Secondly, we assume that the OSN that we want to protect can be represented as a directed graph. Notice that such assumptions are very restrictive in the sense that a significant number of solutions can not be implemented with these constrains. However, we argue that our proof-of-concept become more reliable if the concept of OShN can be proven its effectiveness even under such constrained conditions.

Furthermore, a side objective of our OShN is to minimize the introduced noise. We want to design $G_h$ and $E_b$ in such manner that it allows us to accomplish the previously noted objectives while trying to minimize both the number of nodes $|V_h|$ and edges $|E_h \cup E_b|$ added to the network.

The first goal that an OShN has to accomplish is to be able to attract web crawlers minimizing the time $t_a$. The attraction of a web crawler to our OShN consists of getting that at least one node of $G_h$ is explored by the crawler. This attraction is done by properly selecting the connections of our OShN to the rest of the nodes of the network $E_b$ and by defining the degrees of the exterior nodes of $G_h$.

As we have seen, greedy algorithms select as the next node to crawl the one with the highest degree. For this reason, the node that is being explored at any time tends to have a bigger degree than the previously explored nodes.[1] So when a crawler is launched configured with a greedy algorithm, it will tend to explore the highest degree nodes of the network. Consequently, we would create the set of fake edges $E_b$ between our OShN and the OSN so that they connect $G_h$ with a number $k$ of the highest degree nodes in $G$, ensuring that the crawler will discover those nodes when exploring the highest degree nodes of the network. This accomplish implicitly another goal that is minimizing the annoyances that the OShN cause to users. Since high degree users tend to have thousands of connections with other users, the impact of establishing a connection with $G_h$ is minimum and, in fact, most of the users will not even be aware of this connection.[2] However, the time $t_a$ to attract the crawler using this strategy, depends on the exact greedy algorithm. Note that, when defending the OSN from these particular scheduler algorithms, it is not needed to attach one node of $G$ to more than one node in $G_h$ since all the nodes of $G_h$ connected with the same node in $G$ would be discovered at the same time. However, it may be useful to connect the same node of $G_h$ to many other nodes in $G$ since that would let the crawler discover the node in $G_h$ from different real nodes, reducing the time $t_a$.

Once the attraction has been done, and an exterior node of $G_h$ has been discovered, we want to maximize the time $t_t$ by forcing the crawler to discover more nodes from $G_h$ and crawling all of them. While the crawler is inside $G_h$, no real nodes are crawled thus no node attributes of real nodes are ever disclosed. However, even when the crawler is inside $G_h$, some real nodes may be discovered, depending on the size of $|E_b|$. Since our OShN is designed towards protecting the OSN from real-degree and unseen-degree greedy algorithms, the best strategy to maximize $t_t$ is to set the degrees of the exterior nodes of $G_h$ to at least $max\{m_i + 1\}$ where $m_i$ is the real degree of their neighbors in $G$. Using this strategy, the time $t_t$ is exactly the time needed for the crawler to crawl all nodes in $G_h$. Then, we can defend the OSN from a crawler by assigning an arbitrary large number of nodes to $G_h$. Notice that trapping indefinitely the crawler in

[1] Note that this is true for the vast majority of starting nodes in the network. Some extreme cases, for instance, starting the crawl in the highest degree node, don't have this property.

[2] Deciding how to deal with uncooperative users is outside the scope of this paper.

$G_h$ imply to assign an infinite number of nodes in $G_h$ which is not feasible in our scenario since we have assumed that our OShN is not dynamic, in the sense that $V_h$ , $E_h$, and $E_b$ remain unchanged during the execution of the crawler.

There are many possible configurations that meet the above requirements. For instance, we can design $G_h$ as a complete graph of $d$ nodes where all nodes have degree $d-1$ except for $v_h^0 \in V_h$, which has degree $d$. The additional edge incident to this node is going to be our bridge edge $e_b^0 = (v^0, v_h^0) \in E_b$, which will link our honeynet $G_h$ with the real graph $G$. As we want to ensure that the crawler is not able to escape from the honeynet until it has crawled all the nodes inside $G_h$, we will force that interior nodes of $G_h$ have a higher degree than the node that has served as an entry point to the honeynet $v^0$. For this reason, we will set $d = max\{m_i + 2\}$, so the interior nodes of $G_h$ will have one more link than the most connected node of $G$. Notice that doing so, the entry node $v_h^0$ has exactly the degree of $v^0$ plus 2. Even though a 1 point degree increment will be enough to force the crawler to crawl $v_h^0$ just after crawling $v^0$, incrementing it by 2 allows us to construct $G_h$ in an easy manner, avoiding having to spend computational resources in the design of $G_h$. Figure 1 shows an example with $m_i = 3$.

## 5.2 Experimental Results

We have simulated the correctness of our OShN over the Flickr OSN, taking as a testbed the data collected by Mislove *et al.* in [9] which contains over 11 millions of users. This dataset is one of the most complete OSN data available and can be used as a testing set for OSN analysis. We have centered our experiments in the Flickr network, for which this dataset contains almost the 27% of nodes existing on the network at the time of the crawl $(1,846,198$ nodes) with its relations $(22,613,981$ links). Our experiments are done considering that our OSN graph $G$ is exactly the Flickr graph that had been retrieved in [9]. The diameter of this graph $G$ is 27, the radius is 13 and its mean degree is 12.24. The highest degree of a node in $G$ is $26,185$.

Since real-degree greedy is the scheduler algorithm used as a base point for the tests in [5], we have conducted our experiments with a crawler configured with this algorithm as a scheduler. Two termination conditions have been set for the crawler to stop his job: to reach 1,000 crawled nodes, $n_{crawl} = 1,000$, or to crawl the $v_h^0$ node, which would be the first node in $G_h$ that has been crawled. Furthermore, another end crawling condition has been added when there are no $V_{disc}$ nodes left to crawl, in case the initial seed belongs to an isolated component of the graph containing less than 1,000 nodes.

Assuming these settings, we have created our experimental OShN by generating a complete subgraph of $d = 26,187$ nodes, such that every node in the $G_h$ has exactly degree 26,186 except for a node $v_h^0 \in V_h$, for which we set a degree of 26,187.

We have conducted $18,461$ experiments (1% of the total number of nodes in the data testbed) in order to evaluate the attraction and trapping capacity of our OShN. For each experiment, we select a random node in the Flickr network and we launch a crawler using this node as initial seed and the configuration detailed above. In 12,283 of the conducted experiments, the 66.53%, the OShN was able to attract the web crawler and the crawler crawled the gateway node $v_h^0$. For these experiments, the crawler only need 5.09 hops (in mean) to reach $v_h^0$ from the initial seed. This value indicates that the time $t_a$ for this proof-of-concept is really low. The leaked information obtained by the crawler is very low, since, in mean, only 5 nodes are crawled by the crawler and the mean number of discovered nodes is 12,645.60 nodes, that is less than the 0.7% of the entire network. Notice, however, that the implications of discovered nodes for link privacy are less strong since link information of discovered nodes is incomplete until they are not crawled.

A detailed analysis of the 6,178 experiments where the OShN could not attract the crawler shows that in all cases there is no path between the seed and $v_h^0$. The interesting point is that, for that seeds, the total number of nodes that the crawler is able to crawl is, in mean, 4.40 which imply that the isolated parts of the graph, where the crawler seed has been randomly chosen, are really small.

Obviously, regarding the design of the OShN, the trapping time $t_t$ was maximum, in the sense that the ending condition was met before the crawler left the OShN.

## 6 Conclusions

In this paper, we have studied the effect that web crawlers may have on the information that can be retrieved from an OSN. We review some of the most relevant scheduler algorithms and describe their main properties. We discuss the private information that can be inferred from a social graph, focusing on the communities that can be identified in a graph by means of the connectivity of their members. Then, we analyze the impact of different schedulers algorithms regarding the information that the web crawler retrieve.

All this analysis shows the threat that web crawlers suppose for OSN information. For that reason, and assuming the difficulty to ban web crawlers from OSN, we introduce the concept of online social honeynet (OShN) as a mechanism to achieve some degree of protection against web crawlers. We provide a proof-of-concept of an OShN designed to protect the OSN from a web crawled with a real-degree greedy as a scheduler algorithm. Experimental data shows that the proposed protection is effective and that the amount of OSN data disclosed to the web crawler can be keep at lower levels. Although the proposed OShN only protects the OSN from a specific crawler configuration, it requires low $|E_b|$ values, which makes it easy to be implemented in real world environments.

We have provided some hints towards the construction of the honeynet graph and some of the conditions that force the the crawler to enter the honeynet once it has been discovered and that ensure that the crawler is not able to exit the honeynet once it is inside. However, a detailed analysis on the construction of the honeynet graph remains to be done. The exact construction of a graph that meets the requirements needed for the honeynet while minimizing the overhead introduced to the network is an interesting future work to proceed with.

Moreover, an interesting feature to require to our OShN is that it is not distinguishable from the rest of the network. In doing so we assure that web crawlers can not detect when they are inside the OShN.

On the other hand, in our discussions, we have assumed that our OShN is static in the sense that elements in $V_h$, $E_h$, and $E_b$ remain unchanged during the execution of the crawler. However, a dynamic model can present some advantages to both the effectiveness of the protection as well as the resources used to hold the OShN. Designing such OShN is further work to be done.

**Acknowledgements.** This work was partially supported by the Spanish MCYT and the FEDER funds under grants TSI2007-65406-C03- 03 ”E-AEGIS”, TIN2010-15764 ”N-KHRONOUS”, and CONSOLIDER CSD2007-00004 ”ARES”.

## References

1. Heydon, A., Najork, M.: Mercator: A scalable, extensible web crawler. World Wide Web 2, 219–229 (1999)
2. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems 30, 107–117 (1998)
3. Shkapenyuk, V., Suel, T.: Design and implementation of a high-performance distributed web crawler. In: Proc. of the Int. Conf. on Data Engineering, pp. 357–368 (2002)
4. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: Ubicrawler: a scalable fully distributed web crawler. Softw. Pract. Exper. 34, 711–726 (2004)
5. Ye, S., Lang, J., Wu, F.: Crawling online social graphs. In: Proceedings of the 2010 12th International Asia-Pacific Web Conference, APWEB 2010, pp. 236–242. IEEE Computer Society, Washington, DC, USA (2010)
6. Korolova, A., Motwani, R., Nabar, S.U., Xu, Y.: Link privacy in social networks. In: CIKM 2008: Proceeding of the 17th ACM Conference on Information and Knowledge Management, pp. 289–298. ACM, New York (2008)
7. Gjoka, M., Kurant, M., Butts, C.T., Markopoulou, A.: A walk in facebook: Uniform sampling of users in online social networks (2009)
8. Krishnamurthy, B., Gill, P., Arlitt, M.: A few chirps about twitter. In: WOSP 2008: Proceedings of the First Workshop on Online Social Networks, pp. 19–24. ACM, New York (2008)
9. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: IMC 2007: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp. 29–42. ACM, New York (2007)
10. Backstrom, L., Dwork, C., Kleinberg, J.: Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In: WWW 2007: Proceedings of the 16th International Conference on World Wide Web, pp. 181–190. ACM, New York (2007)
11. Hay, M., Miklau, G., Jensen, D., Weis, P., Srivastava, S.: Anonymizing social networks. Technical report (2007)
12. Zheleva, E., Getoor, L.: Preserving the privacy of sensitive relationships in graph data. In: Bonchi, F., Ferrari, E., Malin, B., Saygın, Y. (eds.) PInKDD 2007. LNCS, vol. 4890, pp. 153–171. Springer, Heidelberg (2008)

13. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: 2008 IEEE 24th International Conference on Data Engineering, pp. 506–515. IEEE, Los Alamitos (2008)
14. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 93–106. ACM, New York (2008)
15. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: SP 2009: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, pp. 173–187. IEEE Computer Society, Washington DC, USA (2009)
16. Pérez-Solà, C., Herrera-Joancomartí, J.: OSN: When multiple autonomous users disclose another individual's information. In: International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing, pp. 471–476. IEEE Computer Society, Fukuoka (2010)
17. Wasserman, S., Faust, K.: Social network analysis: methods and applications. In: Structural Analysis in the Social Sciences, vol. 8. Cambridge University Press, Cambridge (1994)
18. Lee, K., Caverlee, J., Webb, S.: The social honeypot project: protecting online communities from spammers. In: Proceedings of the 19th International Conference on World wide web, WWW 2010, pp. 1139–1140. ACM, New York (2010)