CERIAS Tech Report 2011-08
3-Clique Attacks in Online Social Networks
by Rahul Potharaju, Bogdan Carbunar, Cristina Nita-Rotaru
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

3-Clique Attacks in Online Social Networks

Rahul Potharaju†, Bogdan Carbunar‡, Cristina Nita-Rotaru†
† Department of Computer Science, Purdue University, IN, USA
‡ Pervasive Platforms and Architectures, Motorola Labs, IL, USA
Email: rpothara@purdue.edu, carbunar@motorola.com, crisn@cs.purdue.edu

ABSTRACT

Online Social Networks (OSNs) have become ubiquitous in the past few years, counting hundreds of millions of people as members. In this paper we show that the ease of accessing third party information by engineering OSN features, makes users vulnerable to infiltration attacks. Providing invaluable user context information, such attacks can become dangerous tools in the hands of spammers and phishers. Using a set of primitive attacks, we formalize a new infiltration attack called the *3-Clique* attack. We design an automated attack system, iFriendU, to demonstrate the effectiveness of these attacks on more than 10,000 Facebook users. We show that the *3-Clique* attack outperforms any existing attack by at least 75% in the number of users it can befriend. We propose a novel OSN security framework, called MORPH-x to defend against infiltration attacks. We show the effectiveness of our solution through extensive simulations on a large Facebook social graph. We prove its practicality by implementing MORPH-x as a web application and demonstrate user interest through a user study. We show that our solution imposes only negligible computing overheads on its users and succeeds in blocking the studied attacks in 93-98% of the cases.

1. INTRODUCTION

Online Social Networks (OSNs) such as Facebook have become ubiquitous in the past few years, counting hundreds of millions of people as members. OSNs allow users to form friendship relationships, join groups, communicate and share information with friends. Most OSN users are likely to be well behaved. However, the amount and ease of accessibility of personal information (e.g., date of birth, location, status updates) available on such sites is likely to draw a wide range of users with a malicious intent. Available information can be used by malicious users to launch spamming and phishing attacks. Most of these attempts have financial gain as their ultimate goal. Spammers send out mass advertisements to a large number of users in hopes of selling their products. Phishers, on the other hand, attempt to fraudulently acquire sensitive information from a victim by impersonating a trusted third part (e.g. a banking corporation). In a study by Gartner [2], about 19% of all those surveyed reported having clicked on a link in a phishing email, and 3% admitted to giving up financial or personal information. This reasonable yield, despite having little information about the target victim, suggests that the effect can be more serious when additional victim information is available.

In this paper, we show that an attacker can obtain such information by infiltrating OSNs, using Sybils – fake accounts controlled by the attacker. To this end, we identify several attacks and formalize a novel *3-Clique* Attack, to establish and leverage common context with victims in order to infiltrate even tightly knit communities. We propose an attack system, called iFriendU, a back-end driver written in Java to control Mozilla Firefox through Javascript code injection. We use iFriendU to demonstrate the seriousness of the *3-Clique* attack on more than 10,000 Facebook users over a period of 6 months. Our experiments show that the *3-Clique* attack can be easily automated, easy to perform and efficient – up to 78% of targets fall victims, exceeding by 75% the effectivenss of existing attacks. Moreover, we show that persistence pays off – repeating the same attack may turn rejects into accepts and that sharing more friends with the victim increases the attacker's success rate.

While public key certificates issued by trusted entities could be used to mitigate this problem, the tedious and costly registration process is likely to act as a deterrent to most OSN users. Moreover, a variety of defenses against Sybil attacks in P2P systems [13, 23, 31, 32] have been proposed in the recent years, relying on the assumption that Sybils cannot establish an arbitrarily large number of social connections to non-Sybil nodes [25]. One of the conclusions of the iFriendU-based attack mentioned above is that this assumption does not hold in the context of OSNs thus, rendering these defenses inapplicable in our context.

Instead, we propose and analyze a defense strategy against infiltration attacks in Facebook, that provides user privacy through multiple lines of defense. MORPH-x, the system implementing our solution, works by inferring trust information between users and their friends. We classify trust into two classes: *direct trust*, inferred directly from a user, and *derived trust*, inferred through an action performed by the user, to ensure the flexibility of our defense system. MORPH-x is designed to act as a *security adviser* for a user to answer the question: "*Do I accept this person as my friend?*". It does this by confining the *inviter* to a *probation list*. The inviter is removed from this list only when its trust value, as inferred by MORPH-x, exceeds a certain threshold. Moreover, we propose several strategies for overcoming *cold start* problems, where legitimate users that do not have enough friends may permanently be blocked in probation.

We study the defenses provided by our framework and show that it effectively thwarts the infiltration attacks through extensive simulations on user data collected from 179,000 Facebook users (including 389,000 friendship links). Further, for various MORPH-x user *concentrations*, we show that MORPH-x is able to thwart between 93% (when 10% of all users run MORPH-x) and 98% (when 80% of all users run MORPH-x) of the launched attacks. As a starting point toward proving the feasibility of MORPH-x, we have implemented [1] and stress-tested MORPH-x using 100 PlanetLab [12] nodes. We show that the MORPH-x client overhead is small: to

[1]Please visit http://morphx.info to evaluate MORPH-x

process the initial Facebook account information of a user, we require only 20-40 seconds, even for remote users and during high system loads.

To understand if users would be interested in a system such as MORPH-x, we conducted a small-scale user study. We observed that all the users who participated were enthusiastic about the idea of tagging their friends. Our results show that the social network created by Facebook is different from a real-world social network: *people tend to accept many strangers or untrusted people as friends*.

The rest of the paper is organized as follows. In Section 2 we summarize the properties of social networks on which our work is built and present our attacker model. In Section 3 we present the attacks and in Section 4 we evaluate their effectiveness using our attack system, iFriendU. In Section 5 we introduce MORPH-x and in Section 6 we project its effectiveness and present our implementation. In Section 7 we discuss related work and conclude in Section 8.

2. MODEL

We model the OSN as an undirected graph $G = (V, E)$, where the nodes V represent the registered users and the edges E represent friend relations. We use $e = (u, v)$ to denote the existence of a friendship relation $e \in E$ between two users $u, v \in V$. Without loss of generality, we take the particular case of Facebook but it should be noted that our discussion applies to other OSNs as well. We emphasize the following properties of the social network:

Ease of Registration. Facebook stores an *account* for each registered user. We assume that it is easy for anyone to register a user account. Registering in today's OSNs requires the user to solve a challenge-response CAPTCHA [26] (Completely Automated Public Turing test to tell Computers and Humans Apart). Even though protection schemes like reCAPTCHA (introduced by Ahn *et. al* [27] to complement the weaknesses of conventional CAPTCHAs) and confirmation e-mails may be included, the human costs (money and time) for registering are insignificant. Note however that such protection schemes may make it more difficult but not impossible to massively register accounts.

Friendship and Messaging. Users can establish friendship relations by extending an *invitation*. Once an invitation is sent from a user v to a user u, v's profile is provided to u. User u may accept ("*Confirm*" in Facebook), reject ("*Ignore*" in Facebook) the invitation or leave it pending by not giving a response. If and only if u decides to accept the invitation, is u's profile shared with v. Users also have the ability to send messages to other users, even to non-friends. However, similar to invitations, when a user u sends a message to a user v, u's profile is revealed to v.

Account Data:. As mentioned above, each user has an account, that includes personal profile information, such as name, date of birth, picture, e-mail and snail mail address but also other items such as wall postings, tagged pictures and videos.

Default Access Control Settings:. When a user registers an account it can choose the access permissions to each of its profile's components – who can access which fields of the user's account. While users have the option of changing the default settings, we have noticed in our experiments that many users have not used it.

Attacker Model. We model our malicious user as a person who may be concerned with hiding his or her identity and in addition is a *sensitive information seeker* who tries to invade the privacy of other users. We broadly classify malicious users into impersonators, stalkers, spammers and phishers. An attacker's goal is to collect private user account information from a social networking site (Facebook in our case). We assume an attacker may build or use tools to automate many phases of its attacks. In our model, we also consider the attacker's need for anonymity. For this, we assume an attacker can create and use multiple fake accounts (that do not provide truthful profile information) and may optionally use hijacked or public machines. Thus, in this case, a fake profile is defined to be a profile in which the personal information is vastly missing or different from that of the person owning the profile. This would allow the attacker to maintain its anonymity even in the event that Facebook, upon detecting such stalking activities, would attempt to correlate the fake account's identity to the IP address of the machine used to open the account.

3. INFILTRATION ATTACKS

In this section we identify several attacks that can be launched against Facebook users and formally derive a new infiltration attack called the *3-Clique* attack. We further demonstrate these attacks in Section 4.2.

3.1 Building Blocks

Let M be the attacker, using a fake account to hide its identity. The goal of the attacks is for M to get access to a victim A's account data (mainly its profile information). That is, M wants to change its relationship with A, such that it has access to A's profile data. We assume that initially M has no access permissions to A's profile data. As we will show in Section 4, if an attack fails, M can generate a new fake account and perform other refined attacks.

Candid Attack:. M issues a friend invitation to A. If A accepts it, A and M become friends *i.e.*, A gives explicit rights to M to its profile. Therefore, M learns A's profile. In addition, we observed that A's profile is revealed even if A reacts to the invitation by deciding to reply with a question such as *"Do I know you?"*.

Impersonation Attack:. Deciding the authenticity of profile information is a hard problem. This attacks leverages on this observation and is as follows: M chooses a friend F of A and copies its profile, making it its own. M sends an invite to A with a message of the format "*I have lost my old account and made a new one. Please accept this request.*". Since this behavior can be legitimate, A accepts the invite and reveals its profile. A similar attack was first described in [8].

Chameleon Attack:. Trust is often based on familiarity. This attack builds upon the hypothesis that users tend to accept invitations more easily when they are sent by people with whom they share mutual friends. Then, M initially collects the set of friends of A and issues invitations to everyone or a selected subset (see the *3-Clique* attack discussed next). M then waits for the first of two events: (i) a desired percentage of invitees issue an accept or (ii) a pre-determined time interval lapses (two days in our experiments proved to be sufficient, as shown in Figure 7(b)). M continues with the *candid* attack (against its original target, A). At this point M will have a higher chance of succeeding in becoming A's friend: M and A will likely share mutual friends.

3.2 The 3-Clique Attack

Community infiltration in the context of social networks is a type

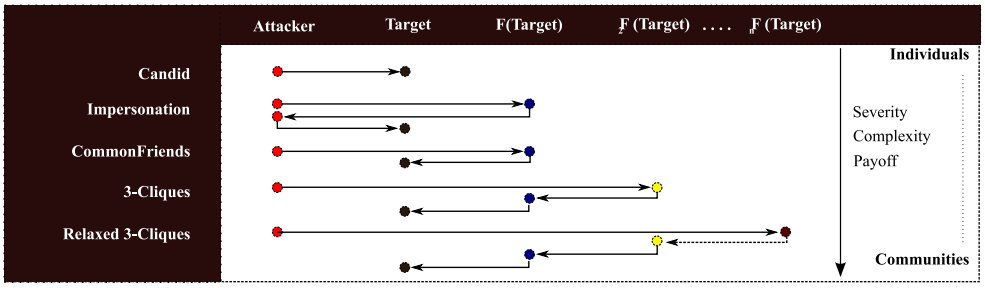
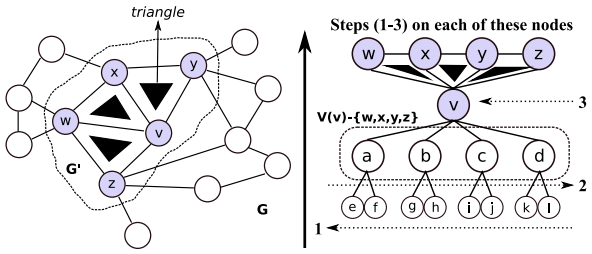


Figure 1: **Attack Map: While the complexity of the attack increases with the attacks in the lower half, the payoff is much higher too. This is evident from the results presented in Section 4.2**



(a) (b)

Figure 2: **(a) Infiltrating a network: Node v has the highest $\delta(v)$ in G', *i.e.*, the attacker's payoff is higher if it establishes a link with v. (b) The *3-Clique* Attack: Infiltration is done in the decreasing order of δ. To establish a link with v (highest δ value) - recursively link with the n^{th} hop network of v in the increasing order of *Social Closeness* to the level above.**

of attack[2] where an attacker targets individuals who are connected together in the form of a community. These attacks could be leveraged for stealing information that belongs to a group. For instance, an attacker who wants to join an employee-only community might have to provide proof of connection with a community member to gain access to the network. In such a scenario, directly attempting to establish a link with the target community member will have a low success rate as will be shown in Section 4. Another instance where community infiltration can be exploited is when launching an *association fallacy* [28] against the entire community. An *association fallacy* is an inductive informal fallacy which asserts that qualities of one user are inherently qualities of another, merely by an irrelevant association. For instance, with enough attackers infiltrating a community, the collective qualities that represent it can be altered. In scenarios like this, the 3-Clique attack can be used to infiltrate communities with a higher chance of success. In the following, we formally define the *3-Clique* attack which can be used to launch such association fallacies and demonstrate its seriousness through a real world experimentation in Section 4.

Definitions: Let a *friendship 3-clique* $\Delta = (V_\Delta, E_\Delta)$ of a graph G = (V,E) denote a subgraph such that $V_\Delta = \{u, v, w\} \subset V$ and $E_\Delta = \{(u, v), (v, w), (w, u)\} \subset E$. Then, let $\delta(v)$ denote the number of friendship 3-cliques of user v. Let the *social closeness* metric, $SC(u, v) = \frac{|F_u \cap F_v|}{|F_u|}$ denote the ratio of the mutual friends of u and v to the total number of friends of u. We define then the *friendship weight* of the link between users u and v to be $w(A, B) = \frac{SC(A,B)+SC(B,A)}{2}$. Intuitively, this is taking into account two factors: what proportion of A's friends are also B's friends and what proportion of B's friends are also A's friends.

[2]Please note the usage of the word 'attack' in our paper. An attack in the true sense would consist of two steps: 1. Inviting a user to accept an invitation, 2. Extracting and storing the user's profile information with a malicious intent. The tools that we have built only perform the first step. They do not allow the collection of any personal information of people that answer the invitation. In fact, an accepted invitation solely incremented a counter.

Algorithm 1 Enhanced Infiltration using 3-Cliques

```
1. G' = Sample(G)
2. 3CSet = get3Cliques(G')
3. DO
4.     v = popUserWithMax3Cliques(3CSet)
5.     firstHopNet(v) = v.friends() − {x : x ∈ G'}
6.     Sort(user ∈ firstHopNet(v), w(v, user), DESC))
7.     FOREACH friend IN firstHopNet(v) :
8.        secondHopNet(v) = friend.friends()
9.        Sort(user ∈ SecondHopNet(v), w(friend, user), DESC)
10.       inviteAll(SecondHopNet(v))
11.    END FOR
12.    Schedule(inviteAll(FirstHopNet(v)), T)
13.    Schedule(invite(v), 2T)
14. WHILE(len(3CSet) > 0)

15. PROCEDURE get3Cliques(G' = (V, E)) :
16.    Setcliques = newSet() :
17.    Construct Friends(user) ∀ user ∈ V
18.    FOREACH relationship IN E :
19.       //relationship consists of user_1, user_2
20.       user_min = min(|user_1.friends()|, |user_2.friends()|)
21.       FOREACH user IN user_min.friends() :
22.             IF(user ∈ (relationship − {user_min}).friends())
23.               store {user, user_1, user_2}
24.             END IF
25.       END FOR
26.    END FOR
27. END PROCEDURE
```

Attack Description:. The *3-Clique* attack is executed using Algorithm 1 and is shown in Fig. 2(b). Let $G' \in G$ denote a subset of the OSN, a tightly knit community that the attacker targets (*line 1*). For each member $v \in G'$, the attacker computes $\delta(v)$ (*line 2*), the *3-cliques* of user v using the method given in (*lines 15-27*). Let v be the member of G' with the highest δ value (*line 4*). The attacker computes the first hop network of v, excluding all the users in G'(*line 5*). Then, the users in this network are ordered decreasingly on the value of their friendship weight to v (*line 6*). An invitation is sent to the second hop network of v (friend-of-friend network) who are again ordered based on their social closeness to each friend of v (*lines 7-11*). Then, after a delay period T, the attacker sends invitations to the first hop network of v (*line 12*). The delay is used to allow the invited second hop network members to accept the invitations. Finally, after another delay period T, the attacker invites the target v (*line 13*). The above process is repeated for all members of G', in decreasing order of their δ values. This is based on the observation that users with high δ values are socially tied to a higher number of groups. Such users may not only be more willing to accept random invitations but more importantly, establishing a friend link with them may further influence other members of G' into accepting the attacker's invitation. An

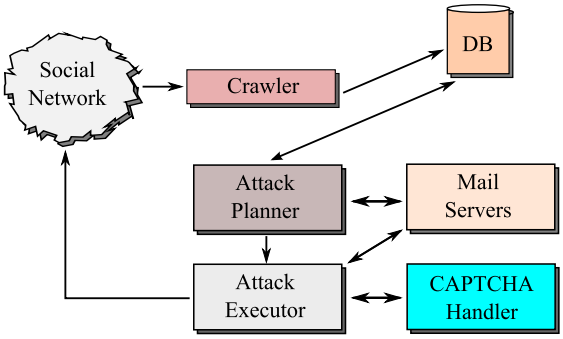


Figure 3: An architectural overview of iFriendU

extension of the attack is what we call the *Relaxed 3-Clique* attack where the attacker can start from the n^{th} hop network of a victim instead of the 2^{nd} hop network as we demonstrated. However, due to space constraints, we will not be discussing this attack further but the attack plan itself is shown in Figure 1.

While this attack is mainly designed for community based infiltrations, it is easy to use this to infiltrate a single user's network by terminating the algorithm after the first iteration.

4. INFILTRATION EVALUATION

In this section, we give a brief overview on the architecture of iFriendU, our attack system and then describe the results of our infiltration attempts using this system.

4.1 iFriendU Architectural Overview

Our prototype attack system [3] relies on an *attack plan* prepared for each of the attacks discussed in Section 3 and is illustrated in Figure 3. The *crawler* component is responsible for crawling the target social networking site and collecting information on users as a seed for the attacks. As Facebook allows anyone to view an arbitrary user's friends list in most cases, we provided our *crawler* with a seed Facebook account using which it recursively crawls the entire network limited by a *depth* parameter customizable through code. We used a 2.4 GHz Intel Pentium 4 with 2 GB RAM to complete crawling of about 50,000 users in less than 5 hours.

The *attack planner* relies on the attack plan (see Figure 1), which is a concise-representation of an attack. For instance, for a *3-Clique* attack, we can specify the order in which the attack execution takes place using "$COM(CLIQUES\ ASC) - L1(SC\ DESC) - L2(SC\ DESC)$" which means, first arrange the the nodes (belonging to the target community) based on their *3-Clique* value. For each node in this order, arrange its 1^{st} hop network in the descending order of the social closeness of each node and so on. The *attack planner* also prepares a list of fake accounts (needed to preserve sender anonymity) and a set of targets listed along with

[3]While we have performed our experiments on real Facebook users we note the following. The profiles used in the fake accounts were blank (no other information except a random name), in order to avoid user impersonation. Moreover, only one bit of information has been collected from each user, whether the invitation was accepted or not. We have not extracted or stored any personal information from the users involved and we have not asked users to perform any subsequent actions. Following the experiments, we have shut down all the fake accounts that we have used in the experiment, making any information no longer accessible. We have then sent a debriefing message to all the users involved, specifying that the invitation they have received was part of an experiment, along with the above points as well as an email address where we could be contacted for further questions or concerns.

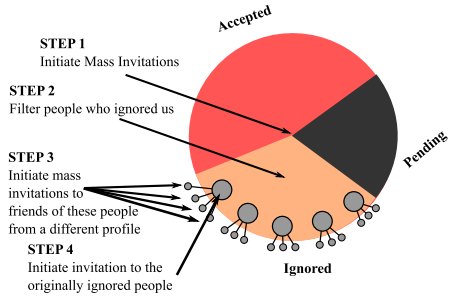


Figure 4: *Chameleon* Attack: Attempt to obtain as many profiles as possible from a set of randomly selected victims.

their friends and proceeds to computing the 3-Cliques by interfacing with a backend MySQL database using Algorithm 1.

The plan generated by the *attack planner* is used as input by the *attack executor*. We have implemented the *attack executor* as a backend driver using Java for Firefox. The *attack executor* launches the browser with itself as the proxy server and then injects JavaScript to execute the *attack plan*. The *attack executor* uses fake accounts to send friend invitations to the target accounts included in the plan. If during its operation the *attack executor* encounters a CAPTCHA, it hands it off to the *CAPTCHA Handler* which uses automated CAPTCHA solvers (e.g., CaptchaBuster [4]) to solve the them. If all attempts fail, the component sends us an email requesting a manual inspection of the CAPTCHA.

Inter-invitation Timing At this point, it is important to mention that the *attack executor* waits for a specific time interval between sending friend requests. The reason for waiting between sending successive invites is that Facebook suspends the account if it sends too many invites. Figure 7(a) shows the results of our experiment when changing the inter-invitation delay time from a *few hundred ms* to *100s*. The y-axis shows the number of invites successfully sent before being banned by Facebook. Note that this number grows quickly but saturates. Our conjecture is that Facebook forbids an account to send more invites when its number of pending invites (sent but not yet answered) exceeds a given value (between 400-500). Because our experiments were designed around sending 1000s of invitations, we extended the *attack executor* to adapt its sending rate depending on its current state. For the first 500 invites, we chose the inter-invite delay randomly between 1 and 15s. For the next 500 invites, we increased the upper limit of the delay to 60s. For the remaining invites, the upper delay limit was further increased to 100s. We were then able to consistently send more than 1500 invites from an account in only a few hours. Note that a fake account can only be used to send a limited number of invites, since the number of pending invites *will* at some point exceeds Facebook's limit.

On a related note, since the fake accounts used by the *attack executor* need to be validated, we setup our own mail server. Both the *attack planner* and *attack executor* interface with the *mail server* for creating accounts as required or for confirming friend requests from other users.

4.2 Infiltration Results

We have launched the following attacks on the Facebook network. As part of the *3-Clique* attack, we collected a relationship graph with 178,000 Facebook users and 339,000 friendship relations to aid us in computing the 3-Cliques of each user.

Chameleon Attack: We illustrate the *Chameleon* attack we implemented, using Figure 4. In the first stage, we selected a random set of 1577 Facebook accounts from our crawled data and launched a *candid* attack using a fake account. Figure 5(a) shows the result of

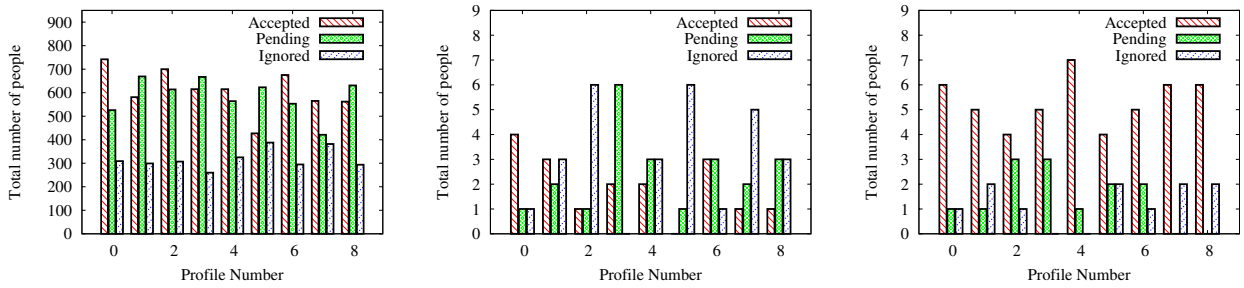


Figure 5: (a) Chameleon Attack Evaluation: Results from the first stage of invitations to the 1577 users (b) Chameleon Attack Evaluation: Results from the second stage of invitations to the 72 users (c) *3-Clique* Attack Evaluation: Last stage of the *3-Clique* attack against the 150, 3-clique members of the 72 target users.

this experiment two weeks after sending the invites: 742 users accepted the invitations, 309 rejected it and 526 were still undecided (pending). From the 309 users who rejected us, we selected a random set of 72 users and sent invitations to all their friends. In order to avoid detection by Facebook, we refrained from sending mass invitations to the friends of these 72 users. Instead, we attached them to 9 different fake accounts, each dealing with 8 out of the 72 users: each fake account is responsible for sending invitations to the friends of 8 of the 72 targeted users ($\sim$ 1600 invites per fake account).

Finally, we use these 9 accounts to complete the *chameleon* attack, by launching a subsequent *candid* attack against the 8 users attached to each of the nine fake accounts. Figure 5(b) shows the result of the *chameleon* attack, again grouped by associated fake account. In total, about 41.7% of the users fell for the attack while 32.8% sustained it. The rest of the 25% were still undecided. Ironically, when we re-sent invitations to the people who rejected us one week later, several have accepted us which indicates that persistence pays off. This is due to a Facebook security glitch that we discovered: flagging a user as *unknown* has no effect. That is, even after being rejected, an attacker can re-send the invite any number of times and it will always be shown as a fresh invite to the victim.

3-Cliques Attack: We have run Algorithm 1 on the relationship graph collected from Facebook (178,000 users and 339,000 friendship relations) and discovered 679,000 3-cliques in less than five minutes (had a similar performance on a different dataset as well [24]). Note that the large number of 3-cliques follows from the small world property of OSNs [29]. To test our *3-Cliques* attack, we targeted the same 72 users used in the *chameleon* attack but after a period of 60 days. Each of the 72 users has the value δ smaller than 500. This is typical for Facebook users, who usually have fewer than 130 friends. In fact, the total number of 3-clique friends for these 72 users was 150 (note that some users participated in multiple *3-Cliques*).

We executed the steps specified in the *3-Cliques* attack description (see Section 3 against these 150 users (which form the target community G‘). Figure 5(c) shows the result of the *Chameleon* attack, again grouped by associated fake account. The acceptance rate was *79%*. This shows that the *3-Clique* attack is 75% more efficient than the *Chameleon* attack.

Additional Statistics Following our *Chameleon* and *3-Clique* attacks, we have monitored several variables. First, we have measured the number of invites accepted per day, following the beginning of each invite. Figure 7(b) shows our results for 4 out of the 9 fake accounts used to send invites. Most invites are being accepted in the first three days after the beginning of the experiment.

Another metric of interest is the distribution of the number of friends per Facebook account. Figure 7(c) shows the average over the 10000 different accounts targeted in our experiments, split over

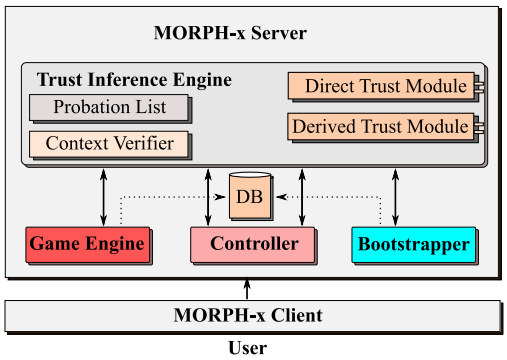


Figure 6: MORPH-x Architecture

three categories: the accounts that accepted, rejected or decided to keep it pending. In the active and rejected groups, most accounts have between 50 and 450 friends. The pending group is more interesting, since most users have fewer than 200 friends. In particular, there are almost 600 users with less than 10 friends. One explanation for this distribution is that some of the accounts that have neither accepted nor rejected our invitations may be inactive. These users have tested Facebook briefly but are not active.

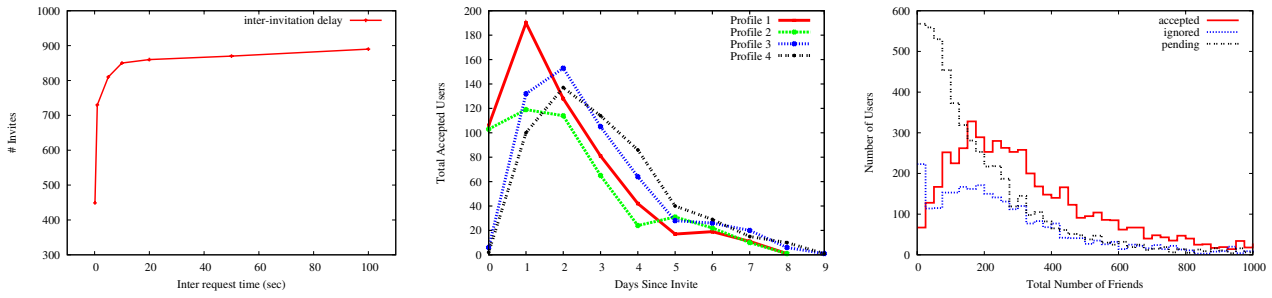
5. MORPH-X SYSTEM DESIGN

We now introduce MORPH-x, a system designed to defend against the social networking attacks previously described. The goal of MORPH-x is to protect rational users, that lack the tools or time to analyze each decision, against cyber stalkers, while minimally impacting the experience of honest social network users.

Solution Overview Figure 4.2 shows the architecture of our system. The client component runs inside a user's account (for instance, a user installed Facebook application). We also provide a trusted third-party server to facilitate user account persistence and infer trust relationships.

Our system is built around the notion of trust, which runs the decision making process. The fact that a user A trusts a user B in some respect informally means that A believes that B will behave in a certain way - perform (or not) some action in specific circumstances [30]. The definition of trust is inherently subjective, and it might not even be feasible to obtain the ground truth so, as a first step, we classify trust into two classes: *direct trust* and *derived trust*. We say there is a *direct trust* relationship between users A and B, if A has a strong reason (knowing B personally or from social gathering etc.) to trust B. A *derived trust* relationship, on the other hand, is learnt through an explicit training process that extracts user feedback regarding trust in its friends.

Overall, MORPH-x works as follows. Whenever user A receives



(a) (b) (c)

Figure 7: Attack Statistics: (a) Capping invitations as a function of inter-invitation delay times (b) Accept rate timeline (c) Friend count distributions of accepted, rejected and pending user accounts

an invitation from B, A's client attempts first to infer whether a *direct trust* relationship exists between A and B. It then places B into the *probation list* of A, where B is denied access to A's profile. We use the *indirect trust* metric to decide when B can be removed from the probation list and promoted to a full friend status, where it is provided with access to A's profile. In the following we describe in detail each of the notions introduced above.

5.1 Inferring Direct Trust

MORPH-x takes advantage of the social component of the design space, where users can be asked to provide personal feedback on other participants. In particular we ask users context verification questions, used to prove the existence of bilateral relationships *i.e.*, prove that a private context is shared with a friend. The *Context Verifier* component of user A generates and sends B a question regarding common context it should have with A (for instance, the place it has met A or details of a discussion it had with A). MORPH-x stores a pre-defined set of contextual questions. It allows each user to choose its preferred questions from the set at install time.

When B answers the context verification question, A's MORPH-x client generates two questions for A: (i) how much it trusts B and (ii) whether the answer to the contextual question correct. For the first question, the client displays a list of trust buttons each named with a keyword ranging from "*No Way!*" (not trusted) to "*Of course*" (very trusted). The user is asked to label B with one of the trust button labels (see Section 5.3 for more details on this question). If A labels B with "Of course" and considers the answer to the contextual question to be correct, the client simply accepts B as a friend. Otherwise, it places B into A's probation list (see next). This approach allows a user that is sure of her friends to bypass the defense mechanisms offered by MORPH-x.

Note that if A and B use smartphones and meet in person, they can use SPATE [15] to exchange authenticated data (e.g., public keys) and later use it to authenticate friend invitations [18]. SPATE authenticated users can immediately by accepted as friends, bypassing MORPH-x.

Shadow Accounts.. The context verification question cannot be sent directly from A's account – this would automatically reveal A's profile to B, (see Section 2) [4]. To prevent this, we define the concept of a *shadow account*. For all practical purposes shadow accounts are fake accounts, whose profile information is impersonal – does not reveal sensitive information. However, unlike attackers our server does mention that these accounts are associated with our service. The MORPH-x server maintains a set of shadow accounts $S = \{S_1, .., S_n\}$. Then, the question previously generated by A's Context Verification component is sent from a randomly selected shadow account $S_r \in S$.

[4] In the experiments described in Section 4 we have seen that a few hundred out of the 10,000 users to which we have sent invites have sent us a "*Hello*" message back, without accepting the invitation – unwittingly revealing their profiles.

5.2 Probation List

If A decides to accept B as a friend (following the interaction with the *Context Verifier*), A's answers to the contextual questions are extracted and used to perform the following actions. If A does not know B or if it marks B's answers to contextual questions as false, the MORPH-x client advises the user to reject the invitation. Otherwise, the client records A's answers on B's context on the server and inserts B into a *probation list*. Let $P(T)$ denote the probation list of T and let users in $P(A)$ be called *probation friends*. Users in $P(A)$ cannot access the profile information of A – B's invitation is left pending. Only when the *derived trust* of B exceeds a pre-defined value, is B promoted to the friend list of A *i.e.* A is advised to accept the invitation. In the following we propose several probation list promotion criteria.

5.3 Inferring Derived Trust

While MORPH-x cannot protect against friends accepted before its installation, it can learn from them and use that information to thwart new attacks. Our approach consists of defining an *Aggregate Trust Criterion* The aggregate trust a user A has in a user B, is based on the social closeness metric of two users (first defined for the 3-Cliques attack (see Section 3)):

$$SC(A,B) = \frac{|F(A) \cap F(B)|}{|F(A)|} \quad (1)$$

The pure use of the social closeness metric in computing trust values is vulnerable to attacks: If B manages to infiltrate a large percentage of A's friends, it accumulates large trust values and may easily be promoted out of the probation list. To counter this effect, we use the following process to evaluate the *trustworthiness* of the social closeness metric:

Training Process - Objective Opinion. We introduce the notion of *Objective Opinion*, $OO(A,B)$, that a user A has in its friend B. The *Objective Opinion* is computed using a training process *i.e.*, through user feedback. We achieve this by deploying a game-like interface, based on the observation (see Burnham *et. al* [9]) that humans have a subconscious friend-or-foe mental approach for evaluating another person.

The training process consists of asking A several types of questions. For a first question type, the game selects between 5 to 10 random friends and displays their photos tagged with their names and a list of *trust buttons*, each named with a keyword ranging from "*No Way!*" (not trusted) to "*Of course*" (very trusted). A is then in-

structed to click on a *trust button* that best describes the current friend – effectively labeling each friend with a trust value. The trust values range from -1 to 1, where the only negative value (-1) is given to the least trusted label ("*No Way*"). If user A has not labeled a friend B, we use the default value $OO(A, B) = 0$. All other trust values are distributed uniformly in the interval $(0, 1]$. If t is the number of trust levels, then the trust values are multiples of $\frac{1}{t}$ ($\frac{1}{t}$, $\frac{2}{t}$, ..., 1).

When enough (or all) friends have been labeled, the system proceeds to ask other types of questions as well. For instance, it selects 2 already labeled friends, the anchors A_1 and A_2, with $OO(A, A_1) = 1/t$ and $OO(A, A_2) = 1/2$. It then selects one unlabeled friend, F, displays the photos of the chosen friends, tagged with their names and asks A to sort them according to their trustworthiness (see the MORPX-x site [7] for the user interface). It then uses the anchors and the relative ranking provided by A to provide tentative trust values for the unlabeled friends. That is, if unlabeled friend F is ranked below A_1, set $OO(A, F) = -1$. If above A_1 but below A_2, $OO(A, F) = 2/t$. If above A_2, $OO(A, F) = 1/2 + 1/t$. In Section 6.2 we provide a detailed description of the how this system has been implemented.

Aggregate Trust Criterion. We now show how the *Social Closeness* and the *Objective Opinion* criteria are aggregated to infer the derived trust of a user A in another user B. If A and B are friends, we define $AT(A, B) = SC(A, B) \times OO(A, B)$. Note that if A has not labeled B in the training process, $AT(A, B) = 0$. Moreover, if A does not trust B, $A(T, B)$ is negative and smaller for higher values of $SC(A, B)$. This enforces our intuition, that if A has many mutual friends with B but does not trust B, it means that B may be an infiltrator or one who has been accepted as a friend due to social obligation. While this discovery comes too late for A (its profile has already been leaked to B) we use this information to protect other users from B.

Specifically, if A and B are not friends, we define the aggregate trust of A and B through their mutual friends. For a mutual friend U, the aggregate trust of A and B is defined to be the product of $AT(A, U)$ and $AT(U, B)$. Then, the aggregate trust of A in B is defined as the sum of the aggregate trusts over all mutual friends of A and B:

$$AT(A, B) = \sum_{U \in F(A) \cap F(B)} AT(A, U) \times AT(U, B) \qquad (2)$$

B is promoted from P(A) when $AT_{(A,B)}$ exceeds a threshold value, T_{AT}. Note that if A and B have no mutual friends, $AT(A, B) = 0$. Moreover, if as mentioned before, A does not trust one of its friends U but it has many common friends with U (U is an infiltrator), the value $AT(A, U)$ will be negative and large in absolute value. Then, $AT(A, U)$ will negate the trust U has in B. Note that it is difficult for an infiltrator U to use this information to play the system. This is because U cannot guess the aggregate trust $AT(A, U)$ – it does not have access to A's opinion.

Handling Cold Starts. The use of the probation list associated with trust levels may introduce a **cold start** problem: legitimate users, that do not have enough friends and thus have a low trust level, may be blocked in the probation lists of all the users they invite. We propose the use of challenges and Merkle puzzles [16] to address this problem. Specifically, a probation friend B, whose trust level does not increase for a pre-defined, system wide parameter, can request the user A, in whose probation list it is blocked, to provide it with computation challenges. For this purpose, A generates Merkle puzzles and sends them to B. If B can solve the puzzles correctly, A increases B's trust level.

An example Merkle puzzle is decrypting an AES cyphertext, encrypted with a small key, while the other bits of the key are fixed and pre-agreed upon. Another Merkle puzzle is inverting a small cryptographic hash value (e.g., given y, find an x such that the last b bits of $H(x)$ and y coincide). Such puzzles assume that B has to perform a brute force attack, thus consume a quantifiable amount of CPU cycles. For instance, a 1.6GHz processor, using an OpenSSL implementation, takes 3.6 hours to break a 35 bit AES key and 116 hours to break a 40 bit key. The same processor takes 11.6 hours to invert a 35 bit SHA-1 hash and 373.3 hours to break 40 bit hashes. The trust level increment should be a function of the difficulty of the solved Merkle puzzle. Long, resource consuming puzzles may discourage their use. We propose instead the use of multiple shorter (1-2 hours) puzzles, each with a smaller trust increment.

This approach is efficient against spammers, as we consider solving many Merkle puzzles to be infeasible. While spammers may consider outsourcing puzzles (e.g., volunteer computing projects or Amazon's mechanical turk), the investment involved may easily exceed the benefits. More information about this is provided on the MORPH-x project page [7].

5.4 Defenses Against Attacks

We now provide an intuition as to why MORPH-x is able to defend against the attacks described in Section 3. In Section 6 we use extensive simulations to validate this intuition. In the following we use M to denote the attacker and we assume that its target A has installed our MORPH-x client.

Candid Attack:. M is accepted as a friend by A only if it is able to generate a correct answer for the context verification question (it needs to know something personal about A which A needs to confirm) *and* M trusts A. Otherwise, A stays under probation until enough trusted friends of A vouch for M. Thus, to infiltrate A, M has to succeed infiltrating and becoming trusted by enough of A's trusted friends (see defense against *Chameleon* attack below).

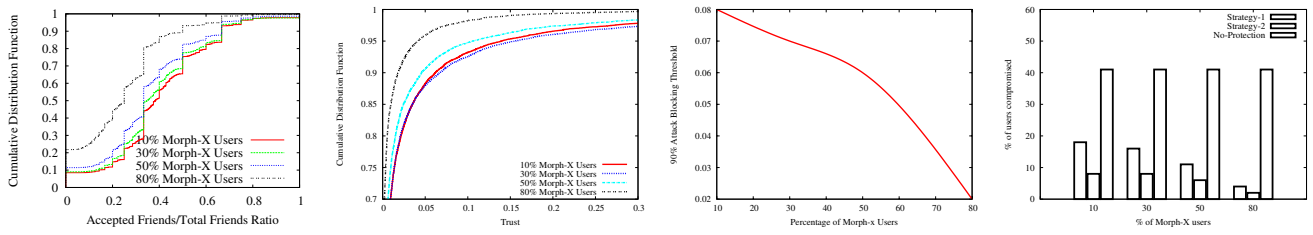
Impersonation Attack: Even though M copies A's profile, when contacting one of A's friends, F (also running MORPH-x), F will label M using M's *user id*, which is a unique identifier assigned by Facebook to a user. Even if M and A have the same profile (including user name) their *user id*s are different. Trust labels are assigned to *user id*s and not to *username*s. Then, F will treat M as just another (random) invitation and not as its friend A.

Chameleon Attack: M needs enough friends of A to accept it as a friend, acknowledge context shared with it and label it with a sufficiently high trust level. Note that all the untrusted friends of A that trust M and all the trusted friends of A that do not trust M will contribute with a negative value to the aggregate trust of A in M. M does not have access to and cannot control the level of trust A has in her friends.

3-Cliques Attack: M needs to infiltrate enough 2-hop friends of a target community member A for it to be promoted into the friend list of enough friends of A for it to be promoted in the friend list of A. If only A runs a MORPH-x client, then the defense proceeds as for the *Chameleon* attack (see above). However, if several of A's friends and of its 2-hop friends have MORPH-x installed, M's chances of infiltration are further reduced. If M is unable to infiltrate those nodes, the *aggregate trust* of A in M will be even smaller.

6. MORPH-X AS A REAL TOOL

To gain a better understanding on the performance of our system we have evaluated its following aspects (i) *Effectiveness*: How



(b) (c) (d)

Figure 8: (a) CDF of Trust(T,I) in MORPH-x using the *Social Closeness* criterion when launching a *3-Clique* attack against 2100 users. (b) CDF of Trust(T,A) in MORPH-x using the *Aggregated Trust* criterion when launching a *3-Clique* attack against 2100 users. (c) Evolution of T_{AT} for MORPH-x users employing *Aggregate Trust* promotion criterion to block 90% of the attacks for various MORPH-x user *concentrations*. The decrease is sub-linear, with T_{AT} as small as 0.02 when 80% of the users run MORPH-x. (d) Comparison of the effectiveness of 2 MORPH-x strategies with non-MORPH-x users in blocking *3-Clique* attacks.

fast can MORPH-x thwart attacks and how long does it take to build trust values for a user's friends?, (ii) *Computational Feasibility*: Can MORPH-x handle a large number of users efficiently?, (iii) *Practicality*: How difficult is it to implement MORPH-x as a real-world system?, (iv) *Deployment*: How easy is it to deploy MORPH-x into the real world social networks? In this section, we use simulations to study effectiveness and an implementation to evaluate computational feasibility and practicality. We also provide preliminary results from a small scale deployment of a subset of MORPH-x.

6.1 Effectiveness

The effectiveness of our system is a function of its user base. Since a large scale deployment requires significant marketing efforts and is outside the scope of this paper, in this section we focus on simulating the effectiveness of MORPH-x in blocking the attacks discussed in Section 3. We have evaluated MORPH-x using data collected from Facebook, consisting of 179,000 Facebook user accounts and 389,000 friendships links amongst them. The simulations have been performed using our custom built social-network simulator whose features include modeling user behavior, friendship links, the ability to send invitations and the ability to install protection systems on user accounts.

In the following, we use a Sybil attacker to launch a 3-Cliques attack against 2100 randomly selected users. We consider three types of strategies: (i) users that run MORPH-x, implementing the *Social Closeness* promotion policy (see Section 5.3), (ii) users that run MORPH-x with the *Aggregate Trust Criterion* promotion policy (see Section 5.3) and (iii) classic Facebook users. We define the *concentration* of MORPH-x users to be the percentage of users that run MORPH-x, out of the population considered. Both MORPH-x promotion strategies use a threshold value as the condition for promotion: T_{SC} is the threshold for the social closeness criterion and T_{AT} is the threshold for the aggregate trust criterion.

Our first endeavor was to investigate the dependence of T_{SC} and T_{AT} on the concentration of MORPH-x users as well as to learn values for the threshold that thwart a high percentage of the attacks. Figure 8(a) shows the CDF for the Trust value, when between 10-80% of users run MORPH-x with the *Social Closeness* criterion. Note that to block 90% of the attacks, the T_{SC} value should exceed 70% *i.e.*, a user should promote a new inviter as a friend only when it shares with it at least 70% of its friends.

Figure 8(b) shows the CDF of the *Aggregate Trust* metric for various *concentrations* of MORPH-x users. Since we cannot simulate the output of the training process, we set the default trust to 1. Note that even for a 10% MORPH-x user *concentration*, a trust value of 0.08 is sufficient to block 90% of the attacks (against MORPH-x users). Figure 8(c) shows the value of the threshold that blocks 90% of the attacks, when the attacks are launched against *concentrations* of MORPH-x users ranging from 10% to 80%. Note that when 80% of users run MORPH-x, a T_{AT} value as small as 0.02 is sufficient to block 90% of the attacks. This is because higher MORPH-x user *concentrations* among a user's friends are more efficient in blocking attacks. Note that MORPH-x users can detect which of their neighbors are also running MORPH-x. This information can be used to locally set a threshold value that provides a desired level of protection.

Figure 8(d) compares the effectiveness of the three strategies, for different MORPH-x user *concentrations*, in blocking the launched *3-Clique* attack. For the MORPH-x users running the *social closeness* promotion criterion, we use a T_{SC} value of 60%. For the MORPH-x users running the aggregate trust criterion, we set the T_{AT} threshold to 0.08. Note that both MORPH-x strategies are much more effective than current Facebook strategy: only up to 16% of the attacks succeed for the first MORPH-x strategy, while only up to 7% of the attacks succeed for the second MORPH-x strategy. This comes in contrast with the current status, where users have no protection system installed, where around 40% of attacks succeed. Note that as the MORPH-x user density increases, the attack success rate decreases significantly: when 80% of the users run MORPH-x's second strategy, only 2% of the attacks succeed.

6.2 Practicality

MORPH-x plays two roles - to allow users to run the training process, and to act as a *security advisor* when a user receives a friend invitation. We now focus on the former, *i.e.* our experiences in designing the training process. We implemented MORPH-x as a Facebook Connect [5] application using Facebook's Graph API [6] which presents a consistent view of the Facebook social graph. A *Connect* application has the primary advantage of being flexible - it is not a Facebook application but rather a social-plugin that makes it a standalone website. As it stands currently, MORPH-x was built using PHP (back-end) and HTML5+CSS3+Javascript (front-end), allowing it to work on Mozilla Firefox and Google Chrome. A snapshot of an early implementation of the system is shown in Figure 9 and can be accessed through the MORPH-x website [7].

Using an application key provided by Facebook, MORPH-x uses the OAuth 2.0 protocol [1] to allow user authentication and authorization. Once the user is authenticated, instructions are provided on how to assign trust values to his friends. This is performed through an interface (shown in Figure 9) that enables the' user to quickly tag a friend (2-4 seconds per friend). We currently provide two training mechanisms: (i) **Tag-O-Rama**, which lets the user assign absolute trust labels to her friends by clicking on one of the buttons under each friend (see Section 5.3), and (ii) **Stack-O-Rama**, which lets the user sort her friends based on a drag-and-drop mechanism. For Tag-O-Rama, the labels on the buttons and their associated objective opinion values are "Of course" (1), "Kind of" (2/3), "I am not sure" (1/3) and "No way!" (-1). 0 was used for the objective value when the user did not tag a friend. In addi-

tion, the interface displays the social closeness for each displayed friend, through a graphical meter which we call the *Trust-O-Meter*, situated on the right for easy reference. The described system is fully operational on an Intel Core 2 Duo processor with 2048 MB RAM and can be accessed through the URI given in [7].

Our on-going work includes designing an intuitive *security advisor* interface for MORPH-x. Specifically, we are addressing the challenge of designing an interface that imposes minimum restrictions on the user without compromising on the defense it offers. We would also like to emphasize that we are working on a client-only solution as opposed to MORPH-x's client-server architecture to address any scalability issues that may arise. We are doing this by off-loading the computational overhead of getting user information etc. to the client itself and storing data locally. Since it is well beyond the scope of this paper to describe these implementation details comprehensively, we have chosen to concentrate our efforts only on the client-server architecture of MORPH-x instead.

6.3 Computational Feasibility

We have used 100 PlanetLab [12] nodes, spread all over the world, to simultaneously access the MORPH-x website. Our goal was to evaluate the performance of MORPH-x under stress. Figure 10(a) shows the result of this experiment for three cases where the user logs in (i) for the first time, (ii) for the second time with its browser cache disabled, (iii) for the second time with browser cache enabled. The response time is higher the first time a user loads MORPH-x, requiring between 20-40 seconds. We emphasize that these values are measured when 100 users distributed all over the world *simultaneously* login for the first time. The large variation is due to variations in the per-user number of friends that need to be loaded from the user's account, to variations in Facebook's instantaneous load and to the latencies experienced by remote users. However, for subsequent logins, the response time is significantly reduced to 3-5 seconds.

As a next step, we have studied the performance overhead incurred by the server-side modules when building and maintaining probation lists – using the (more expensive) *Aggregate Trust* criterion. Figure 10(b) shows the time taken for this operation, as a function of the number of user accounts (ranging from 100 to 50,000, collected from Facebook servers when a new user installs MORPH-x). Even for 50,000 users, maintaining the probation list takes only around 30 seconds. Note that the probation lists need to be (iteratively) updated only when a new friendship relation, involving one of the maintained users, is established. Given that this operation is not time sensitive, friendship updates can be batched, allowing the probation list update process to be performed only at fixed intervals.

6.4 Deployment

We have conducted a small-scale user study to gauge the user reaction to a system like MORPH-x. We primarily advertised by word-of-mouth and email. In total, 22 out of the 30 people we contacted agreed to participate in this study. The task of the user in this experiment is to use the training mechanism of MORPH-x to tag their friends. 70% of the people had more than 200 friends and were enthusiastic about tagging them despite the long lists. 10% suggested that we provide them either with more *trust* labels or less ambiguous ones because they had trouble mainly with the *"I'm not sure"* label. For this reason, we re-ran the pilot study on all the users using a new set of labels *"Maybe Yes"* instead of *"Kind Of"*, *"Maybe No"* instead of *"I'm not sure"*. The results of this study are shown in Figure 10(c) for a subset of the users. The implications are two-fold:

(i) **Disjunction:** Participants had a long list of friends that they did not trust. This we observed irrespective of the set of labels we used. In fact, in our second run, this became even more clearer as the list of untrusted friends got transferred into the *"Maybe Not"* category. This result could be hinting that the friend circle projected by Facebook is quite different from the real life network. The more surprising result is assignment of the "*No Way!*" label to a number of friends despite finding them in the friendlist.

(ii) **Feasibility**: Even a simple tagging mechanism can be used to build trust values. Most users were enthusiastic about the user study despite having no incentives for completing the study. Thus, with the correct tagging mechanism, this clearly shows the role that a system such as MORPH-x can play in today's OSNs in acting as a user's own personal security advisor.

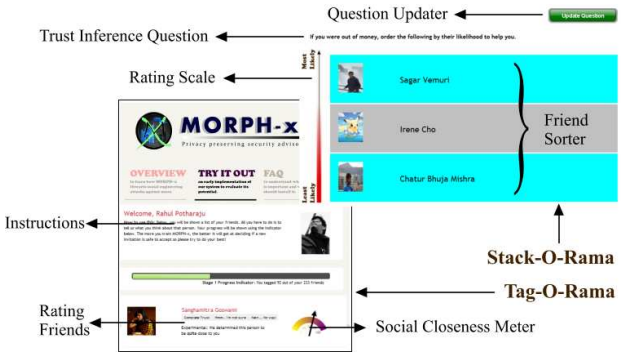


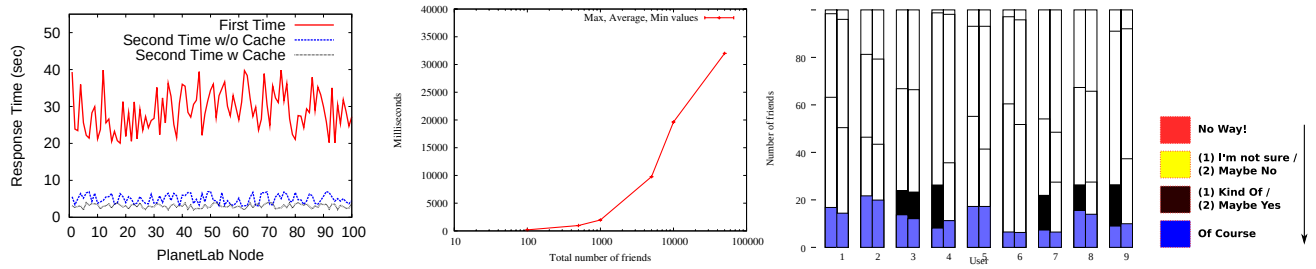
Figure 9: Implementation of MORPH-x

7. RELATED WORK

Recently, OSNs have started being rigorously studied by the scientific community. Bilge et al. [8], study solutions for collecting personal user profiles from various OSNs, including Facebook. They conjecture and prove that people are more willing to accept friend requests from people they already know. They devise an impersonation attack which they test on 700 users. While no defense mechanisms were proposed, in our work we design MORPH-x, whose goal is to defend against a suite of profile collection attacks, including impersonation attacks (see Section 6) Jagatic et al. [14] conducted a phishing study to establish a baseline for individual phishing attacks. The percentage of victims who disclosed their personal information to a phishing site underscores the need for developing phishing prevention techniques. In our work, we establish a baseline for passive invitations where only an offer to establish friendship is sent to a user without an actual message thus emphasizing the success rate achievable through simple random actions..

Tootoonchian et al. [22] propose Lockr, a system for improving the privacy of social networks. It achieves this by using the concept of a social attestation, which is a credential proving a social relationship. Lockr further uses zero knowledge proofs to prevent reuse of such credentials and various secure multi-party protocols to allow users to connect only if certain (private) conditions are satisfied. MORPH-x does not attempt to nor hide personal information from OSN sites, but prevents unauthorized users from accessing personal content, while retaining Facebook's main features.

Caverlee and Webb [11] conducted a large scale study on MySpace, where they discovered interesting patterns, such as high account abandonment rates, language/location correlations and interestingly, that privacy is becoming a concern factor even and mostly for younger users. Nazir et al. [17] conducted a similar study on Facebook, through the development and deployment of three applications that gained significant popularity. The focus of their study is on behavior within a community – in their case, the communities



(a) (b) (c)

Figure 10: **(a) Implementation Evaluation: Overhead of client state collection step, function of number of friends. (b) Implementation Evaluation: Server Side Probation list maintenance overhead. (c) MORPH-x average load times (across 10 runs) as observed from 100 PlanetLab nodes spread across the world. (d) Pilot study of a subset of users who participated in evaluating MORPH-x**

adopting their applications.

Another MySpace study was conducted by Webb et al. [3], to study social spamming. The concept of social honeypots is introduced, which are MySpace accounts created specifically for attracting spam. The results show that social spammers exhibit temporal and geographic patterns, which may be used to automatically detect and even eliminate them. Caverlee et al. [10] continue this work with the proposal of a trust establishment solution for social networks. Trust between two users is defined to be a factor of the quality of the interaction between the two users. MORPH-x differs in that is uses a concept that fits the social network model: a game-like Facebook application that collects direct user feedback on the trustworthiness of friends.

Singh et al. [20] study a Facebook privacy issue based on the observation that Facebook applications have unrestricted access to the accounts of the users that install them, leaving users vulnerable. They propose a framework for developing social networking applications that preserves user privacy by enforcing a complete mediation of all communications between applications and external entities. In our work we focus on preventing different types of Facebook privacy leaks e.g., Facebook community exploration, sending massive invites and so on (see Section 3 for a list of attacks we propose and consider in our work).

Sirivianos et al. [21] proposed FaceTrust, a system relying on social tagging as a mechanism for establishing the credibility of users and the trustfulness of their assertions. While our work could use the mechanisms proposed in [21], we note that it is unclear whether users would indeed take advantage of such mechanisms and what incentives could be provided for them to be truthful (e.g., for them to say that indeed a friend is lying). Baden et al. [19] proposed BondBreaker, a Facebook game where users can establish "bonds" by asking questions and "break" existing bonds by guessing the answer. This approach is similar to the direct trust mechanism used by our context verification component. Note however that since for many question types the answer space is small, this approach is vulnerable to Sybil attacks: Sybil accounts attempt to answer questions randomly until one guesses the correct answer. This motivates the use of a probation list with different associated promotion criteria.

8. CONCLUSION

In this paper we propose and study privacy attacks that can be deployed against online social network users and introduced a novel *3-Clique* attack for community infiltrations. Through an extensive implementation we show that such attacks are very efficient in extracting user information that can be very valuable to spammers and phishers. We design and implement MORPH-x, a solution for protecting this information and sharing it only with trustworthy friends. We show that MORPH-x imposes small computational overheads and efficiently defends against the attacks identified.

9. REFERENCES

[1] Oauth 2.0 protocol. `http://tools.ietf.org/html/draft-ietf-oauth-v2-10`.

[2] Gartner study finds significant increase in e-mail phishing attacks. `http://www.gartner.com/press\_releases/asset\_71087\_11.html`, April 2004.

[3] Social Honeypots: Making Friends with a Spammer Near You. In *Proceedings of the 5th CEAS*, 2008.

[4] Captcha buster. http://captchabuster.com, 2010.

[5] Facebook connect. `http://developers.facebook.com/docs/guides/web`, 2010.

[6] Facebook Graph API. `http://developers.facebook.com/docs/api`, 2010.

[7] Morph-x implementation. `http://morphx.info`, 2010.

[8] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All Your Contacts are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *Proceedings of WWW*, 2009.

[9] T. Burnham, K. McCabe, and V. Smith. Friend-or-foe intentionality priming in an extensive form trust game. *Journal of Economic Behavior & Organization*, 2000.

[10] J. Caverlee, L. Liu, and S. Webb. Socialtrust: tamper-resilient trust establishment in online communities. In *Procs. of JCDL*, 2008.

[11] J. Caverlee and S. Webb. A large-scale study of MySpace: Observations and implications for online social networks. In *Proceedings of the AAAI*, 2008.

[12] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An Overlay Testbed for Broad-Coverage Services. *SIGCOMM CCRev.*, 2003.

[13] G. Danezis and P. Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *Proceedings of NDSS*, 2009.

[14] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Communications of the ACM*, 2007.

[15] Y. Lin, A. Studer, Y. Chen, H. Hsiao, L. Kuo, J. McCune, K. Wang, M. Krohn, A. Perrig, B. Yang, et al. Spate: Small-group pki-less authenticated trust establishment. *IEEE Transactions on Mobile Computing*, 2010.

[16] R. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):299, 1978.

[17] A. Nazir, S. Raza, and C.-N. Chuah. Unveiling facebook: a measurement study of social network based applications. In *Proceedings of IMC*, 2008.

[18] A. Perrig. Private communication.

[19] B. B. Randy Baden, Neil Spring. Identifying close friends on the internet. In *Hotnets*, 2009.

[20] K. Singh, S. Bhola, and W. Lee. xBook: Redesigning Privacy Control in Social Networking Platforms. In *Proceedings of 18th USENIX Security Symposium*, 2009.

[21] M. Sirivianos, K. Kim, and X. Yang. Facetrust: assessing the credibility of online personas via social networks. In *Proceedings of HotSec*, pages 2–2, 2009.

[22] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: Better Privacy for Social Networks. In *Proc. of ACM CoNEXT*, 2009.

[23] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *Proceedings of NSDI*, 2009.

[24] B. Viswanath, A. Mislove, M.Cha, and K. Gummadi. On the

Evolution of User Interaction in Facebook. In *Procs. of WSON*, 2009.

[25] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based sybil defenses. In *Procs. of SIGCOMM*, 2010.

[26] L. Von Ahn, M. Blum, and J. Langford. CAPTCHA project. 2006.

[27] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465, 2008.

[28] D. Walton. *Ad hominem arguments*. University Alabama Press, 1998.

[29] D. Watts and S. Strogatz. Small world. *Nature*, 1998.

[30] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems-a distributed authentication perspective. In *Proceedings of IEEE SnP*, 2002.

[31] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *In Procs. of the IEEE SnP*, 2008.

[32] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: Defending against sybil attacks via social networks. *SIGCOMM CCRev.*, 2006.