On the Runtime Enforcement of Evolving Privacy Policies in Online Social Networks*

Gordon J. Pace[1], Raúl Pardo[2], and Gerardo Schneider[2]**

[1] Dept. of Computer Science, University of Malta, Malta.
[2] Dept. of Computer Science and Engineering,
Chalmers | University of Gothenburg, Sweden.
{gordon.pace@um.edu.mt, pardo@chalmers.se, gersch@chalmers.se}

Abstract. Online Social Networks have increased the need to understand well and extend the expressiveness of privacy policies. In particular, the need to be able to define and enforce *dynamic* (and *recurrent*) policies that are activated or deactivated by context (events) or timeouts. We propose an automaton-based approach to define and enforce such policies using runtime verification techniques. In this paper we discuss how our proposed solution addresses this problem without focussing on concrete technical details.

1 Introduction

Online Social Networks (OSNs) are not only a way to keep in touch and socialise but a way of life. Nearly 70% of the Internet users are active on OSNs as shown by a recent survey [5], and this number keeps increasing. New technologies usually comes with a lot of opportunities, but also with new sometimes unexpected threats and challenges. One of such problems in OSNs is that of privacy. Very often users' requirements are far from the privacy guarantees offered by OSNs which do not meet their expectations [6].

OSN privacy policies can typically enforce many desirable policies; for instance, in Facebook users can state polices like: *'Only my friends can see a post on my timeline'* or *'Whenever I am tagged, the picture should not be shown on my timeline unless I approve it'*. Many other policies, however, are not possible to enforce, although they might be important from a user's privacy perspective. For instance, users cannot specify privacy policies such as *'I do not want to be tagged in pictures by anyone other than myself'*, nor *'Nobody apart from myself can know my child's location'*. These limitations to what current privacy control settings can describe and enforce might be limiting user adoption and use of effective privacy policies.

* Published in *7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation - ISoLA'16 (2); Track: Runtime Verification and Enforcement, the (industrial) application perspective*, volume 9953 of LNCS, pages 407-412. Springer, Oct 2016. DOI: 10.1007/978-3-319-47169-3_33

** Corresponding author.

Besides, the current state of the art in privacy settings do not take into account the *dynamic* aspect of privacy policies. That is, privacy policies should consider the fact that the networks *evolve*, as well as the privacy preferences of the users. An OSN may evolve in different ways, by introducing new users, by sending posts and invitations to participate in events, by accepting such invitations, liking pictures and posts, etc.

The privacy policy may also evolve due to explicit changes done by the users (e.g., a user may change the audience of an intended post to make it more restrictive), or because the privacy policy is dynamic *per se*. Examples of the latter, are for instance: *'My boss cannot know my location between 20:00-23:59 every day'*, *'Only my friends can know my location from Fridays at 20.00 till Mondays at 08:00'*, and *'Co-workers cannot see my posts while I am not at work, and only family can see my location while I am at home.'* These are recurrent policies triggered by some time events ("every day between 20:00 and 23:59", and "every week from Friday at 20.00 till Monday at 08:00"), or location-based ("not at work [...] at home"). Other policies may be activated or deactivated by certain events: *'Only up to 3 posts, disclosing my location, are allowed per day in my timeline'*.

In this paper, we are concerned with evolving privacy policies of the latter kind: how to define and enforce dynamic (possible *recurrent*) privacy policies that are activated or deactivated by context (events or location) or timeouts. In particular, we aim at proposing one approach on how to define such policies and discuss how to guarantee at runtime their enforcement. We do not develop in this paper a concrete technical solution but rather outline an automata-based approach, and discuss how it could be implemented.

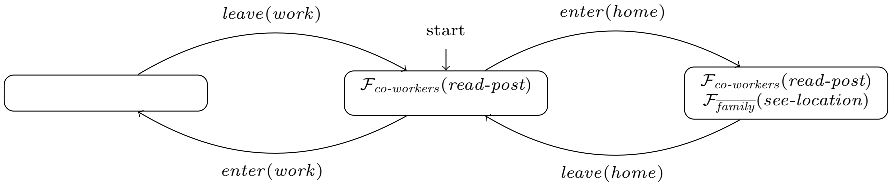
2 An Approach to Represent Evolving Privacy Policies

An evolving policy effectively corresponds to temporal modalities sitting above predicates in a static policy logic. Given that much work has been done in the area of representing and enforcing static policies, with different approaches being proposed to address different contexts, we have chosen to keep the approach policy logic agnostic and have a description of which policies are triggered as time progresses.

The formalism we propose to describe the evolving behaviour of policies is to use deterministic automata with transitions labelled by events which the policy's environment — in our case the online social network — can perform. By tagging each state with a static policy which expresses what is and what is not allowed, we automatically obtain an operational view of which policies are switched on and off during the system's lifetime. By synchronising the policy automaton's state with the events from the social network, other users and the general policy environment, we can add and remove policies appropriately.

For example, consider the policy *'Co-workers cannot see my posts while I am not at work, and only family can see my location while I am at home'* (P1). If we use the static policy operator $\mathcal{F}_g(x)$ to denote that anyone in group g

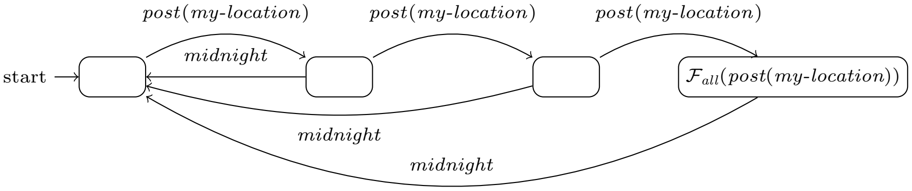
is forbidden from performing action x (x can refer to posting, viewing a post, liking a post, etc.) we can express the policy while not being at work to be $\mathcal{F}_{\textit{co-workers}}(\textit{read-post})$, and the policy when not at home to be $\mathcal{F}_{\overline{\textit{family}}}(\textit{see-location})$ (we use $\bar{g}$ to denote the complement of a group of users g). By synchronising with the actions of our social network application registering our arriving at and leaving a location (*enter*(l) and *leave*(l) respectively), we can express the evolving policy in the following manner:



$leave(work)$
start
$enter(home)$
$\mathcal{F}_{\textit{co-workers}}(\textit{read-post})$
$\mathcal{F}_{\textit{co-workers}}(\textit{read-post})$
$\mathcal{F}_{\overline{\textit{family}}}(\textit{see-location})$
$enter(work)$
$leave(home)$

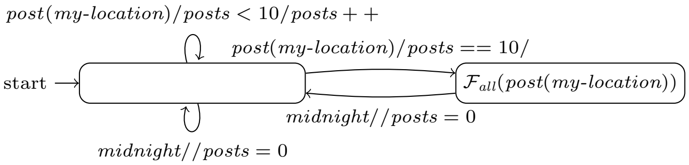
This approach allows the adoption of any static policy language and allow its dynamic extension. In addition, we can easily extend definitions of concepts such as policy refinement and policy conflicts already defined on static policies to the policy automaton level — for instance, an evolving policy represented as a policy automaton P is a refinement of another evolving policy P' if, for any trace t, the policy in the state of P reachable after following trace t is a refinement of the policy in the state of P' reachable from following trace t.

In practice, when specifying these policy automata, we allow for symbolic states to be used by using variables which can be checked and updated on the transitions. For instance, consider the policy *'Only up to 3 posts, disclosing my location, are allowed per day in my timeline'* (P2), which can be encoded as the following automaton:



$post(\textit{my-location})$
$post(\textit{my-location})$
$post(\textit{my-location})$
$midnight$
start
$\mathcal{F}_{all}(post(\textit{my-location}))$
$midnight$
$midnight$

If the maximum number of posts were to be increased, specifying this as an explicit automaton can quickly become unwieldy, and using a symbolic state variable can simplify specifying the automaton in a more concise way. In the representation below, each transition is labelled as: *event*/*condition*/*state-update* — triggering when the specified event happens and the condition holds, performing the state update before proceeding. The property allowing for 10 location posts can be expressed in this notation in the following manner:


$post(my\text{-}location)/posts < 10/posts++$
$post(my\text{-}location)/posts == 10/$
start
$\mathcal{F}_{all}(post(my\text{-}location))$
$midnight//posts = 0$
$midnight//posts = 0$

Note that this can be reduced to an infinite state explicit automaton with the state consisting of a tuple (q, n) where $q \in \{1, 2\}$ represents the two states, while $n \in \mathbb{N}$ represents the value of the variable *posts*.

3 Runtime Enforcement of Evolving Privacy Policies

One of our objectives is to have an effective enforcement mechanism for evolving privacy policies based on policy automata in a real OSN. In this section, we give an overview of how such enforcement can be achieved.

Using policy automata to model the evolution of the privacy policies would make it possible to define a modular enforcement of evolving policies. As mentioned in the previous section, policy automata are independent of the static policy language of the OSN, and consequently, they are also independent of the underlying enforcement of each particular static policy. Therefore, the two main required ingredients for an enforcement of evolving privacy policies are:

i) An OSN with a built-in enforcement for static privacy policies,
ii) A tool which monitors the evolution of the OSN and controls the state of the policies at each moment in time.

All the popular OSNs such as Facebook, Twitter, Google+, Instagram, etc. have a built-in mechanism for enforcing static policies. However, monitoring the events occurring in the OSN requires full access to the internals of the system, which is normally not publicly available for those OSNs. Thus, in order to have a working implementation we should target an open-source OSN, like for instance the distributed OSN Diaspora* [3]. Pardo and Schneider [8, 7] have recently extended Diaspora* with a prototype implementation of some privacy policies defined in the $\mathcal{PPF}$ framework [4]. $\mathcal{PPF}$ is a formal (generic) privacy policy framework for OSNs, which needs to be instantiated for each OSN in order to take into account the specificities of the OSN. $\mathcal{PPF}$ was shown not only to be able to capture all privacy policies of Twitter and Facebook, but also more complex ones involving implicit disclosure of information.

The remaining element of the enforcement is a tool which is able to model policy automata (cf. ii)). In previous work Colombo *et al.* introduced LARVA [2], a tool to automatically generate a monitor from properties expressed in DATEs (*Dynamic Automata with Events and Timers*). Though the expressiveness of DATEs is not sufficient to encode policy automata, we believe they can be extended in order to reach the intended expressiveness.[3]

[3] All the behaviour and information in DATEs are carried on the transitions: states are only used as a way to define transitions.

In order for the runtime enforcement to work we would need to use a communication protocol between Diaspora* and Larva. Every time that a relevant event occurs in Diaspora* it should be reported to Larva. Then Larva would update the state of the privacy policies (if applicable), and whenever a privacy policy is updated Larva would report this change to Diaspora*, which would update the corresponding (static) privacy policy (see Fig. 1).

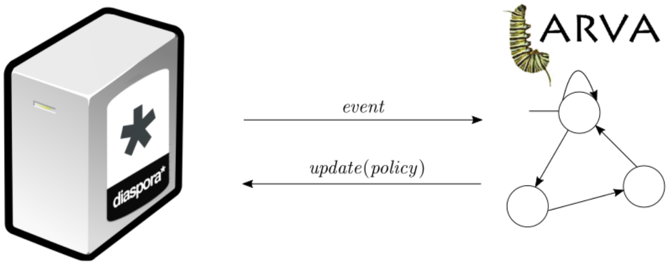


Fig. 1. High-level representation of the Diaspora*-Larva communication

One possibility is to implement the communication protocol using sockets. Imagine we were to implement the policy (P2) described in the previous section, which states that at most 3 times per day posts containing a user's location are allowed. Every time that a user publishes a post including a location, a message would be sent to the Larva monitor. This message must include the information of which users are included in the post, since their location is (potentially) going to be disclosed. At this moment, Larva would update the state of the policy automaton by either increasing the variable controlling the number of posts including a location (for each user) and/or updating the state of the automaton. Finally, if the automaton goes to a state in which no further user location disclosures are allowed, then Larva would communicate with Diaspora* updating the relevant privacy settings for the user, and Diaspora's built-in mechanism for enforcing static policies would then take care of the rest. At midnight, the automaton returns to the initial state, updating the static policy in Diaspora* accordingly, i.e., permitting location disclosure once again.

4 Conclusions

We have sketched an approach to formally represent evolving privacy policies, and a practical solution enabling the synthesis of monitors to enforce such policies. Our objective here is not to provide a technical solution, but rather to give some initial ideas on how to address it and to pave the way to further research on the topic.

We are currently looking into the formal definition of policy automata, and a translation into LARVA monitors [2, 1] instantiated for Diaspora*. In particular, we would like to apply the approach by using privacy policies using an instantiation of the $\mathcal{PPF}$ privacy framework [8].

Acknowledgements This research has been supported by: the Swedish funding agency SSF under the grant *Data Driven Secure Business Intelligence*, the Swedish Research Council (*Vetenskapsrådet*) under grant Nr. 2015-04154 (*PolUser: Rich User-Controlled Privacy Policies*), and the European ICT COST Action IC1402 (*Runtime Verification beyond Monitoring (ARVI)*).

References

1. C. Colombo, G. J. Pace, and G. Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *FMICS'08*, volume 5596 of *LNCS*, pages 135–149. Springer-Verlag, 2009.
2. C. Colombo, G. J. Pace, and G. Schneider. LARVA — Safer Monitoring of Real-Time Java Programs (Tool Paper). In *SEFM'09*, pages 33–37. IEEE Computer Society, 2009.
3. Diaspora*. https://diasporafoundation.org/. Accessed: 2016-07-01.
4. $\mathcal{PPF}$ Diaspora*. Test pod: https://ppf-diaspora.raulpardo.org. Code: https://github.com/ raulpardo/ppf-diaspora. 2016.
5. A. Lenhart, K. Purcell, A. Smith, and K. Zickuhr. Social media & mobile internet use among teens and young adults. millennials. *Pew Internet & American Life Project*, 2010.
6. Y. Liu, K. P. Gummadi, B. Krishnamurthy, and A. Mislove. Analyzing facebook privacy settings: User expectations vs. reality. In *ACM SIGCOMM IMC'11*, pages 61–70. ACM, 2011.
7. R. Pardo. *Formalising Privacy Policies for Social Networks.* Department of Computer Science and Engineering, Chalmers University of Technology, 2015. Licentiate thesis.
8. R. Pardo and G. Schneider. A formal privacy policy framework for social networks. In *SEFM'14*, volume 8702 of *LNCS*, pages 378–392. Springer, 2014.