Caprice: A Tool for Engineering Adaptive Privacy

Inah Omoronyia[1], Liliana Pasquale[1], Mazeiar Salehie[1], Luca Cavallaro[1], Gavin Doherty[3], Bashar Nuseibeh[1,2]

[1] Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland
[2] Department of Computing, The Open University, UK
[3] Lero – The Irish Software Engineering Research Centre, Trinity College Dublin, Ireland

ABSTRACT

In a dynamic environment where context changes frequently, users' privacy requirements can also change. To satisfy such changing requirements, there is a need for continuous analysis to discover new threats and possible mitigation actions. A frequently changing context can also blur the boundary between public and personal space, making it difficult for users to discover and mitigate emerging privacy threats. This challenge necessitates some degree of self-adaptive privacy management in software applications.

This paper presents *Caprice* - a tool for enabling software engineers to design systems that discover and mitigate context-sensitive privacy threats. The tool uses privacy policies, and associated domain and software behavioural models, to reason over the contexts that threaten privacy. Based on the severity of a discovered threat, adaptation actions are then suggested to the designer. We present the *Caprice* architecture and demonstrate, through an example, that the tool can enable designers to focus on specific privacy threats that arise from changing context and the plausible category of adaptation action, such as ignoring, preventing, reacting, and terminating interactions that threaten privacy.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques
General Terms
Design, Security, Human Factors, Management
Keywords
Privacy, adaptive software, changing context, selective disclosure

1. INTRODUCTION

As software applications become increasingly ubiquitous and dynamic, users privacy requirements change and become more difficult to manage [2]. One class of such requirements is *selective disclosure* – deciding what information to disclose, in which context, and the degree of control an individual has over disclosed information. The pervasive nature of software applications means that users need to be continuously aware of new and changing operational context of their applications, and to understand the implications of disclosing personal information in new contexts. Also, when context changes frequently (e.g., changing time, location and activities) then the boundary between public and personal spaces can also get blurred [3] and introduce unexpected privacy threats. In such scenarios, users may be unaware of when and for what purpose sensitive information about them is being collected, analysed or disseminated. This makes it even more difficult for users to adapt their application to continue to satisfy their privacy requirements.

This challenge calls for a more systematic approach to enable the explicit consideration of privacy in the engineering of critical software applications. Firstly, it is essential to continuously examine context changes, such as changing spatio-temporal user attributes, as well as the environmental or regulatory constraints over which such attributes are disclosed. Secondly, such applications should be able to reason over changing context to discover privacy threats, and take actions for their mitigation. Although there are some methods for addressing privacy at design time [6], they do not target privacy threats arising from changing context, nor the adaptation countermeasures that should be triggered by applications when such threats occur.

In this paper, we present *Caprice*, a tool aimed at supporting software engineers in the design of applications that appropriately adapt their behaviour to mitigate privacy threats. At design-time, this tool provides software engineers with some insights about the functional behaviour of the system under development and runtime context changes that can threaten privacy. The core features of *Caprice* include: (1) identifying contextual properties to be monitored in order to detect context changes that might threaten privacy; (2) reasoning over a history of agent interactions to discover privacy threats, and (3) suggesting possible threat mitigation actions based on the severity of the discovered threat. In this paper, we demonstrate Caprice using a mobile application that enables a group of runners to share running data.

The next section contains some background on our approach. Section 3 describes the *Caprice* system architecture, while section 4 illustrates *Caprice* and its user interface by using an example scenario. Related work and conclusions are in sections 5 and 6, respectively.

2. OUR APPROACH

Consider a designer of a privacy critical system that has a set of privacy requirements to meet, and is given a set of privacy policies and a domain model representing the operational context of the designed system. One way of satisfying such privacy requirements in a system design is to justify that the disclosure of information by the system as a result of an information request, does not result in the violation of associated privacy policies. The focus of *Caprice* (http://caprice.codeplex.com/) is therefore to help designers discover possible privacy threats resulting from information disclosure and potential mitigation actions. Specifically, *Caprice* is meant to inform an implementation of adaptive privacy systems. *Caprice* is potentially useful for supporting privacy-by-design by making explicit the context of

ASE'12, September 3–7, 2012, Essen, Germany

privacy violations and possible mitigation actions at design time. The underlying research upon which *Caprice* is based is described elsewhere [1].

The approach implemented in *Caprice* consists of three analysis steps. First we identify the set of attributes that need to be monitored to detect context changes that may threaten privacy. This is important as monitoring all attributes that characterise a usage context can incur performance cost. This step is enabled via the notion of *privacy awareness requirements* – the set of attributes that need to be monitored to detect context changes that may threaten privacy. This involves the parameterisation of attributes in a domain model to instantiate an operational context. Then, by relating the system behaviour to privacy policies we identify a subset of attributes in the operational context to be monitored.

In the second step, based on the monitored attributes, we carry out privacy threats analysis to discover operational context that can violate privacy. Caprice makes use of Barth *et al.'s* contextual integrity framework [5] to justify the preservation or violation of privacy in a system's behavioural model. Contextual integrity posits that the transfer of information about a subject from a sender to a receiver, in a specific context, is tied to certain transmission principles, such as notice, consent, and confidentiality. In *Caprice*, we operationalize these transmission principles by using privacy policies. Then, based on the behavioural representation of the system and a domain model, we reason over the operational contexts that threaten privacy. The reasoning is an iterative process that simulates a possible valuation/aggregation of attributes in an operational context that can result in a privacy policies violation.

For the final step, by computing the severity of a discovered threat, a possible adaptation action is suggested to the designer. The severity of a threat is computed based on a utility value derived from the sensitivity and obfuscation levels of the disclosed attribute. The sensitivity level of an attribute describes its importance to its owner (the subject). The obfuscation level refers to the precision/accuracy of the attribute being disclosed. For example, the parameterised attribute *age = 55* is more precise compared to *age = 45-60.* Thus, the severity of a threat will be lower if the sensitivity level is very low and the obfuscation level is inaccurate and imprecise. Conversely, threat severity will be higher if the obfuscation level is accurate and precise. *Caprice* suggests four categories of adaptation actions based on the severity of the discovered threat. These categories include Ignore, React, Prevent or Terminate. A user can choose to *ignore* a threat if the expected severity is low (i.e. utility value is high). Conversely, a higher than expected severity level might require a different action. In *react*, a user can allow the message to be transferred, but additional conditions need to be satisfied by the sender/receiver to mitigate the effects of discovered threat. *Prevent* involves a user simply objecting to the message transmission between the sender and the receiver. Finally, *terminate* is the action that is selected when threat severity is at the peak level. This involves stopping further message transfers, thus withdrawing from associating with the group objective.

3. CAPRICE SYSTEM ARCHITECTURE

Caprice is implemented using the Microsoft .Net framework and its architecture consists of three layers, as shown in Figure 1. The modelling layer (layer 1) generates the domain, policy and behavioural model of the system. The domain model is instantiated using an interpreter from a domain knowledge repository. The policy model retrieves policy statements from a policy repository. Finally, a behavioural model is generated and represented as an FSM of the system.

The second layer is composed of the operational context emulator, the FSM-Policy connector, and the agent interaction simulator. An agent here represents the sender, receiver or subject of transferred information. The operational context emulator evaluates a sequence of operational contexts based on attributes defined in the domain model. The FSM-Policy Connector overlays state transitions with privacy policies. Using the agent interaction simulator, *Caprice* can then simulate interaction between multiple agents. This is achieved by associating an FSM instance with each agent. Then a random message transfer involving agents is simulated using a Monte Carlo simulation algorithm. It is also possible for the designer to customize the policies associated with each FSM instance representing an agent.

In the third layer, for every operational context simulated in the second layer, the designer is presented with a runtime view of FSM instances. This includes the possible privacy threats and mitigation actions that can be generated by the added operational context. The privacy awareness engine filters an evaluated subset of monitored attributes from the operational context. Subsequently, the privacy threats reasoner checks if the operational context of an agent interaction satisfies the privacy policies of the associated subject. It does so by first building a model of knowledge gained by interacting agents about the subject over a sequence of interactions. Based on LTL properties associated with the ensuing policy, the privacy threats reasoner then checks if the modelled knowledge will satisfy the privacy policies of the subject for that operational context. Finally, if the policies are not satisfied, the Mitigation Action Analyser recommends a possible mitigation action based on predefined adaptation rules.

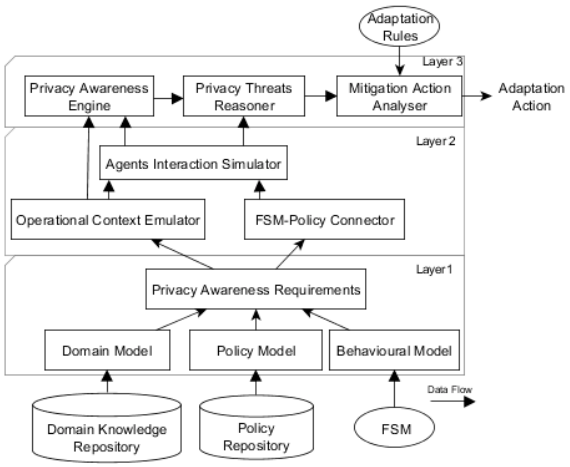

Figure 1 Caprice architecture

4. USING CAPRICE

In this paper, we use a track sharing mobile application for runners and other outdoor activity to illustrate the usage of *Caprice*. Similar examples of such application include B.iCycle (http://b-icycle.com/), MyTracks (mytracks.appspot.com/) etc. Typically, such applications enable a group of runners to share live GPS tracks and performance statistics with fellow runners and other agents such as their fitness instructors and physicians. For this example, privacy management includes the capability of runners to decide the limits of information disclosure to other agents – about their current location, running time, distance, age, heart rate, burned calories, weight loss, etc. Effective adaptive

privacy requires runners to understand information flows, weigh the consequences of sharing information, and make informed, context-specific decisions to disclose or withhold information.

The overall workflow for using *Caprice* separates between three tasks. This includes the process of instantiating the domain and behavioural models with associated policy repository in order to identify privacy awareness requirements. The second task is related to discovering privacy threats. The last step involves the suggestion of plausible adaptation actions to mitigate the threats discovered. In this section, we describe how these three tasks can be performed in *Caprice*.

4.1. Privacy Awareness Requirements

The identification of privacy awareness requirements starts with identifying attributes that characterise the domain of system operation. This is an activity that can be carried out by a domain expert/system designer. The FSM of the designed system is then modelled by identifying the states, events, and transitions that define the behaviour of the system. In particular, an FSM description includes highlighting the domain attributes that are disclosed as a result of a state transition, as well as privacy policies that constrain defined state transitions. The set of attributes that need to be monitored for privacy threats analysis is the subset of domain attributes that are common to both the set of disclosed attributes resulting from a transition, and privacy policy that constrains that transition.

Domain attributes are captured by clicking the 'add attribute' button on the domain model tab of *Caprice*. This step also allows the definition of inference relations amongst attributes. An inference relation is a phenomenon that enables the deduction of previously unknown information from another disclosed attribute. In *Caprice*, this is achieved either via direct implication or aggregation. They both involve the use of established rules that predict the value of an attribute to some degree of accuracy. Implication inference relations are uni/bidirectional relations between two attributes. An example of a bidirectional implication inference is *locationName* $\Leftrightarrow$ *locationCoordinates* (i.e. if a runner's *locationName* is disclosed, it is possible to deduce the *locationCoordinates* and vice versa). Aggregation inference can be deduced by learning patterns that occur in the values of attributes over time. For example, the relation *Weight* $\vDash$ *BMI* $\cap$ *height* infers that the knowledge of a runner's BMI and height may be aggregated to infer the runner's weight. While some forms of aggregation relations can be bidirectional, we have only considered unidirectional aggregations. Generally, these relationships necessitate monitoring additional attributes to satisfy privacy awareness requirements. For example, assuming *Weight* is private for a runner, there is also a need to monitor the disclosure of *BMI* and *height* of the runner. Additionally, a subset of possible attribute values with the associated sensitivity and obfuscation levels are also defined.

The functional behaviour of the system is modelled in *Caprice* using an FSM editor. An example of an FSM is shown in Figure 2 (bottom-middle). For every event that is created, the domain attributes that characterise the event are identified. For example, the event *EstablishFix* enables the application to obtain an initial GPS fix, and the attribute that characterises this event is the runner's current *locationCoordinates*. Subsequently, for every state transition that is defined, the FSM editor enables a designer to select the event that triggers the transition. We envisage that not all states or transitions in a state machine will be involved in the disclosure of privacy sensitive information. In such cases, a designer can uncheck the affected transitions or states.

The key assumption in associating privacy policies with state transitions is that the transition from one state to another is bound by specific transmission principles operationalized as privacy policies. Thus, privacy policies that are associated with attributes disclosed as a result of the event are automatically selected as the

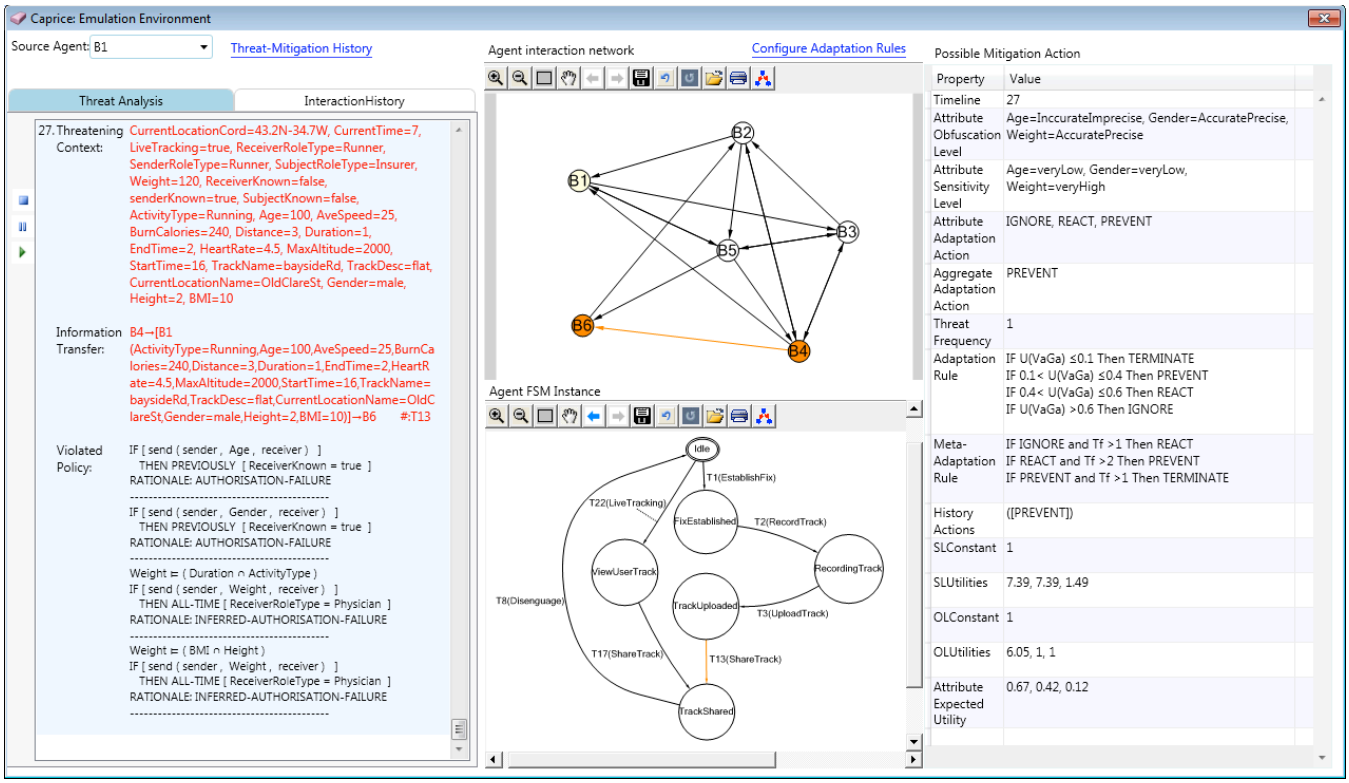

Figure 2 Privacy threats analysis and mitigation action selection in Caprice

transmission principles for that transition. As Figure 2 (bottom-left) shows, privacy policies are expressed using IF-THEN-UNLESS statements tagged with temporal constraints. These temporal constraints can be expressed in the past (using PREVIOUSLY and LAST-TIME temporal operators), in the future (using NEXT-TIME, HENCEFORTH, EVENTUALLY temporal operators), and in all states (using ALL-TIME operator).

4.2. Privacy Threats Analysis

Privacy threat analysis is an automated process that is triggered by selecting a source agent in the *Caprice* environment, and clicking the 'play' button. The analysis involves the discovery of different operational contexts (a specific set of valuations of attributes in a domain model – example in Figure 2 top-right) over which if a state transition occurs, then the associated transmission principles will not be satisfied. In order to achieve this, *Caprice* models users as a group of interacting social agents. These agents perform actions involving personal information in a given operational context. Each agent is represented by an FSM instance that is used to model its behaviour across a sequence of operational contexts. A state transition is then triggered when an agent interacts with another agent, by requesting or receiving information. Each agent can adjust privacy policies associated with its FSM instance based on specific preferences. Agents also share knowledge among their group members and keep memory of past interactions. In this way, a subject's decision to consider a specific information request or response as privacy threatening is based on two factors: the history of operational context and what knowledge other agents in the group already have about the subject. We model such group knowledge using epistemic modal reasoning that is based on the S5 axiomatic system [7]. Figure 2 (top-middle) shows an example of simulated interactions involving 6 agents (B1-B6), where B4 transfers previously acquired information about B1 to B6. The left corner of Figure 2 shows the operational context that threatens privacy, as well as the information about B1 that is transferred from B4 to B6, and the violated privacy policies.

4.3. Mitigation Action Selection

Once a privacy threat is identified, *Caprice* recommends a mitigation action based on predefined adaptation rules configured by the designer. These rules are based on severity and frequency of discovered threats. The severity of a threat is calculated using a utility function. Figure 2 (right) shows a suggested mitigation action based on the privacy threat resulting from interaction between B4 and B6. For this example, the age, gender and weight of B1 is being disclosed to B6 in a privacy threatening context. Subsequently, a *Prevent* adaption action is suggested to the designer in order to mitigate the inappropriately disclosed of information related B1. In this illustration, both sensitivity and obfuscation levels have the same utility weight, and the partial utility functions are positive and negative exponential functions with a damping factor of 3 and 2, respectively.

5. RELATED WORK

We are not aware of any similar work for engineering adaptive privacy. General work on inconsistency management in software engineering has considered so-called "repair actions" [8] to mitigate discovered inconsistencies. However, despite some similarities, it does not address privacy and its dynamic context sensitivity. On the other hand, Spiekermann and Cranor provided a framework for engineering privacy [6], but without focusing on the challenges brought by changing context nor on software engineering concerns. Our research brings these two perspectives together. It also differs from traditional requirements monitoring approaches, such as those proposed by Fickas and Feather [9], which do not address monitoring privacy-critical contextual properties.

6. CONCLUSION AND FURTHER WORK

This paper has demonstrated *Caprice* – a tool to aid the design of privacy-critical systems. The focus has been on systems whose intended usage is characterised by a dynamic environment where context changes frequently. We demonstrated a simulation environment in *Caprice* that discovers privacy threats and suggests plausible mitigation actions. Assuming an attribute is being transferred from a sending to a receiving agent (expressed as a transition in a state machine), *Caprice* reasons over what associated agents may know over time about the subject of their transmissions.. If such knowledge will violate the subject's privacy policy, a mitigation action is recommended to the designer. These actions are based on the nature of disclosed information, and the frequency of occurrence of the threats. The demonstration highlights the plausibility of *Caprice* to support designers in making informed decisions about what privacy management capabilities to enable in software systems.

Further work will focus on the semantics of the different categories of mitigation actions and the details of how they can be implemented and applied to the system. Furthermore, we have only considered agent interactions within a single group, future work will focus on extending *Caprice* to analyse interactions between multiple groups.

6. ACKNOWLEDGMENTS

This work was supported, in part, by SFI grant 10/CE/I1855 (for CSET2) to Lero, Microsoft SEIF Award (2011) and the European Research Council.

7. REFERENCES

[1] I. Omoronyia, L. Pasquale, M. Salehie, G. Doherty, B. Nuseibeh. Engineering Adaptive Privacy: A requirements-driven approach for mitigating mobile privacy threats, technical report, Lero-TR-2012-03, 2012.

[2] C. Mancini, K. Thomas, Y. Rogers, B. Price, L. Jedrzejczyk, A. Bandara, A. Joinson, and B. Nuseibeh. From spaces to places: Emerging contexts in mobile privacy. 11th Int. conf. on Ubiquitous comp., 2009, Orlando, Florida, USA.

[3] S. Lahlou, M. Langheinrich and C. Röcker, "Privacy and trust issues with invisible computers," Commun. ACM, vol. 48, no. 3, pp. 59-60, 2005.

[4] A. Acquisti, and J. Grossklags, "Privacy and Rationality in Individual Decision Making," IEEE Security and Privacy, vol. 3, no. 1, pp. 26-33, 2005.

[5] A. Barth, A. Datta, J. Mitchell, and Nissenbaum, H., Privacy and Contextual Integrity: Framework and Applications. In Proc of the IEEE Sym. on Security and Privacy. IEEE Computer Society, Washington, DC, USA, 184-198, 2006.

[6] S. Spiekermann, and L. F. Cranor, Engineering Privacy, Sof. Eng., IEEE Trans on,.35(1) 67-82, 2009.

[7] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, Reasoning about Knowledge. MIT Press, 1995.

[8] C. Nentwich, W. Emmerich, and A. Finkelstein. 2003. Consistency management with repair actions. ICSE '03. IEEE Computer Society, Washington, DC, USA, 455-464.

[9] S. Fickas and M. S. Feather, "Requirements Monitoring in Dynamic Environments," 2nd Int. Sym on Req. Eng., 1995