PRIVACY PRESERVING SURVEILLANCE AND THE TRACKING-PARADOX

Simon Greiner[1], Pascal Birnstill[3], Erik Krempel[3],
Bernhard Beckert[2] and Jürgen Beyerer[3]

[1] *simon.greiner@kit.edu*
Karlsruhe Institute of Technology (KIT),
Institute of Theoretical Informatics,
Am Fasanengarten 5, 76131 Karlsruhe (Germany)

[2] *beckert@kit.edu*
Karlsruhe Institute of Technology (KIT),
Institute of Theoretical Informatics,
Am Fasanengarten 5, 76131 Karlsruhe (Germany)

[2] *prename.surname@iosb.fraunhofer.de*
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB,
Dept Secure Communication Architectures,
Fraunhoferstraße 1, 76131 Karlsruhe (Germany)

Abstract

Increasing capabilities of intelligent video surveillance systems impose new threats to privacy while, at the same time, offering opportunities for reducing the privacy invasiveness of surveillance measures as well as their selectivity. We show that aggregating more data about observed people does not necessarily lead to less privacy, but can increase the selectivity of surveillance measures. In case of video surveillance in a company environment, if we enable the system to authenticate employees and to know their current positions, we can ensure that no data about employees leaves the surveillance system, i.e., is being visualized or made accessible to an operator. In contrast, due to their lack of computer vision intelligence, conventional video surveillance systems do by design treat each person's privacy equally, independent of whether one has to spend the whole work day under surveillance (e.g. personnel of an airport) or occasionally a limited amount of time (e.g. air passengers). We conceive our approach towards improving the selectivity of video surveillance measures as an interpretation of the principle of proportionality in law.

Keywords: Video surveillance, privacy, security, data protection, formal verification, KASTEL.

INTRODUCTION

Intelligent video surveillance is an active and lively field of research, predominantly in the domains of image exploitation and situation assessment. The availability of privacy-invasive system functionality such as real-time object tracking and automatic extraction of biometric features is becoming reality. Not surprisingly, video surveillance generates an increasing interest among information security and privacy researchers.

A categorical argument against video surveillance targets the chilling effect of such systems, which arguably is in conflict with the fundamental right to free development of the individual. When faced with surveillance cameras, we cannot know whether we are currently observed or not. However, the mere possibility of being observed tends to change the way we behave, which usually is considered an undesired phenomenon in free societies and therefore addressed by legislation. The principle of proportionality, as laid down in articles 8(2) and 52(1) of the Charter of Fundamental Rights of the European Union, demands a careful

weighing of the purpose of a surveillance measure, i.e., the legally protected interest to be defended, against the legitimate interests of people affected by the surveillance measure. However, we do observe that video surveillance is spreading rapidly, even though the proportionality of privacy invasion and utility may not always be justified.

In addition, even if we consider video surveillance to be necessary in particular cases, the question of how and to which extent privacy of the people concerned can be preserved must be evaluated.

Given that modern video surveillance technology works at the level of abstracted objects rather than raw video streams, we argue that the computer vision capabilities of such systems can also be exploited for improving the selectiveness of surveillance measures. Intelligent video surveillance systems are capable of fusing information extracted from video streams into abstracted objects, including attributes such as IDs by face recognition, location, or certain activities. Hence, we can analogously incorporate an authentication mechanism as an information source, which enables the system to determine (group) identities of people that are a priori known to be concerned by the surveillance measure, e.g., employees of an airport as an environment, which is typically equipped with extensive surveillance facilities. If we furthermore assume that the airport operating company trusts in its employees while air passengers should be observed for the sake of civil security, being able to distinguish between airport personnel and air passengers, the system can actively apply privacy-preserving mechanisms on person objects recognized as employees. Such privacy-preserving mechanisms may be applied to video stream visualization (e.g. blurring faces of employees) as well as to abstracted views (e.g. hiding or coarsening positions of employees on an overview map). By this means, we can improve the selectivity of surveillance measures, i.e., employees who have to spend their whole work day in an area under video surveillance can to some extent be relieved from the pressure of video surveillance, while air passenger are being observed as required by the security task.

As stated above, this ability to enforce privacy-preserving mechanisms on particular groups comes at the cost of collecting additional data. In this paper, we investigate how to design modern video surveillance systems that collect certain kinds of data only for the benefit of privacy. Using methods from the area of formal software verification, we show that an according implementation does not expose such data for other purposes. Based on this work we again quote the principle of proportionality and argue that the benefits for privacy outweigh additional information processing, since we ensure that the additional information is never exposed.

RELATED WORK

We concentrate on work on privacy policy enforcement in video surveillance systems, since our work is orthogonal to privacy-enhancing computer vision techniques, i.e., privacy-enhancing technologies for surveillance camera's video streams.

In [1] Senior et al. introduce a privacy-preserving video console for hiding sensitive details in video streams depending on authorization levels. This suggests that the privacy level of exposed video data should be adjusted exclusively to the authorization level of the observer, as opposed to the authorization level induced by the surveillance purpose or by (groups of) observed persons.

Wickrasamuriya et al. enforce privacy policies for video rerendering [2]. Video surveillance is assumed to be restricted to critical regions. Cameras are deactivated by default, yet are activated based on motion detectors detecting people entering such regions. Policies specify access rights to regions and privacy levels for individuals or groups. People are authenticated using RFID tags. When entering a critical region with an RFID tag granting access, one may also be granted a high privacy level, i.e., getting erased from visualized video data. This approach seems to be useful when utilizing video surveillance for observing

people in constrained regions. However, even while staying in the observed area, people can transfer their (group) identity to someone else by passing on their RFID tag.

AUTHENTICATION WITH AN INTELLIGENT VIDEO SURVEILLANCE SYSTEM

In order to enforce (group) identity-based privacy requirements, e.g., hiding employees in the video surveillance process, we need to enable respective persons to authenticate themselves with the system.

We propose to use a two-step authentication scheme using a mobile communication device, e.g., a smart phone or tablet [3]. First, a cryptographic authentication is performed over a wireless network, authenticating the mobile device as belonging to somebody from the group employees (or, as the case may be, a particular person). In the second step the surveillance system replies with a short-lived graphical code, which is easy to recognize for surveillance cameras. When the code is presented to a camera, the authentication as an employee is fused into the associated *guest* object captured by the camera. The object is hence reclassified as an *employee* object and privacy-enhancing mechanisms matching this group identity are triggered.

The association of an object and its (group) identity is maintained by employing the system's tracking capabilities, i.e., keeping track of the position of a person recognized as an employee is crucial for being able to enforce the privacy requirements being due to the group of employees. As stated above, in comparison to a locatable token, this approach has the advantage that it is much harder to transfer ones identity to someone else.

TRACKING-PARADOX

In intelligent video surveillance, we denote the following phenomenon as *tracking paradox*: Assume a video surveillance system that visualizes the positions of guests as pictographs on an abstract area map. Additionally we aim to prevent the target video surveillance system from visualizing, respectively exposing any data about employees in the area under video surveillance. In order to allow for such behaviour, the system needs to track the positions of all objects (including employees) in order to protect the ones that are known as employees.

To understand how this paradox originates, it is important to recap how intelligent video surveillance systems process data. Computer vision algorithms extract information from surveillance cameras' video streams, i.e., feature vectors including the position of the observation, which is then delivered to information fusion algorithms. These algorithms aggregate observations from various information sources, i.e., multiple image or signal exploitation algorithms monitoring the same area, into distinct objects. These objects are then maintained in a data structure, such as a database. A simple example of a fusion algorithm would aggregate all observations within the close proximity of an existing object to this particular object. If no proximal object exists, a new one is created.

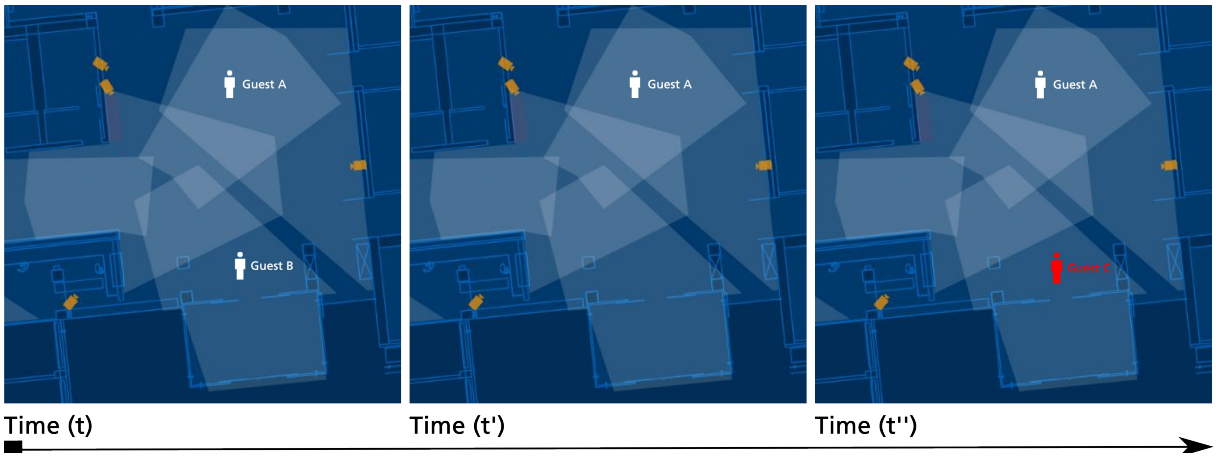

Figure 1: Data representation

Fig. 1 visualizes the state of a surveillance system's database at a given time (t). Currently the system has information about two different objects, denoted as Guest A and Guest B. In the next step (t') the system receives additional information that allows for classifying Guest B as an employee. According to the privacy policy, which forbids tracking of employees, assume that the system now deletes all information about this object. As a result, the object disappears from the map. Therefore, on the first glance, the system seems to adhere to the claimed privacy policy of not tracking employees. However, in step (t'') the tracking-paradox comes into effect. The surveillance system again receives information about an object, which, according to its position, is unknown so far. As there is no object into which the received information can be fused, the system creates a new object called Guest C. Hence, the system is now tracking an employee, even though this employee has just successfully authenticated himself with the system and should be protected.

If we generalize the tracking paradox, we can phrase it as follows: **If the classification into a protected group depends on a subject's private information, then it is impossible to distinguish between private information from protected and non-protected individuals.**

Coping with the tracking-paradox while still fulfilling (group) identity-based privacy requirements, the system's implementation needs to adhere to the following principle: **Collecting a subject's private data for classification purposes is allowed, if and only if it can be shown that it never exposes data of a member of a protected group.** As long as this principle holds, privacy protection for certain groups can be achieved, while others can still be monitored for security reasons.

IMPLEMENTATION

A simplified version of a data store and a fusion algorithm encapsulated in one Java class was implemented. The program maintains a list of employees as well as a list of guests and is capable of tracking the positions of these persons. Additionally, persons are able to authenticate themselves as employees, turning them invisible for the operator. In order to be able to visualize guests in the user interface, observations of such objects can be requested from the system. The signatures of the methods and the fields declared by this class are shown in Fig. 2.

```
1  private int[][] guestVectors;
2  private int[][] coworkerVectors;
3
4  public void updateObservation(int[] observation);
5
6  public int[] getGuest(int pos);
7
8  public void registerCoworker(int[] observation);
```

Figure 2: Signatures of used fields and methods

Two arrays are used to store the features of all objects known to the system. Without loss of generality we simplified the implementation by choosing two 2-dimensional Integer arrays instead of arrays of Object. The array *guestVectors* stores the features of all guests, while *coworkerVectors* hold the information about the employees. Three methods can be used to update the stored objects. The method *updateObservation()* takes as argument a feature vector, which contains the features of an observation as extracted from a camera stream. If the features can be fused with a known guest, the information about this person is updated using the values of the given argument. If no guest fits to the observation, the system checks, if there is an employee suitable for fusion. If neither exists, a new guest is created in the system.

The method *getGuest()* returns the features of a guest at the given index, or null, if the given index is out of bounds. The method *registerCoworker()* checks, if a guest exists in the system

that fits to the given feature vector. If so, the information about this guest is removed from *guestVectors* and added to *coworkerVectors*.

While this implementation is rather simple, it provides all necessary functionality, i.e., its behaviour reflects the behaviour of real surveillance systems on a higher level of abstraction, thus allowing us to analyse it using formal methods. We aim to show that it is possible to implement a data store with a fusion algorithm, which ensures that no information about employees is exposed by the system. By storing more information we want to ensure that less information is exposed to the environment.

VERIFICATION

We analyzed the implementation as described above using self-composition ([7], [8]) in order to proof non-interference properties. In this approach the data in the system is separated into a low and a high part. An environment may learn anything about the system's low values by running a program, but must not learn anything about other values. Formally, two runs of the program are compared, both of which are started in states that agree on the low values, but may differ on the high values. A program satisfies the non-interference property if the low values also agree in the post state. If the environment is now considered to only be able to observe the low values, he cannot learn anything about the high state of the system by analysing the information given to him by running the program.

Allowed information flow conditions and functionality for each public method is specified using JML [9]. A short example of the used JML annotation is shown in Fig. 3. We skip the presentation of the functional part here, since it is out of scope of this paper. Specified low information, which may be known to the environment, includes the features of all stored guests, since they may be shown to the operator. Moreover, features that are extracted from a camera stream may be disclosed under the condition that either the user interface visualizes a guest or the shown person has not registered or been identified as an employee yet, so this person is treated as a new guest.

```
1  public normal_behaviour
2  requires observation.length == NUM_FEATURES && (\forall int i; 0 <= i &&
3      i < guestVectors.length; guestVectors[i] != observation) &&
4      (\forall int i; 0 <= i && i < coworkerVectors.length;
5          coworkerVectors[i] != observation);
6  respects lowvalues, (\exists int j; 0 <= j && j < coworkerVectors.length;(
7      (observation[POS_X] - coworkerVectors[j][POS_X]) < BLURX &&
8      (-1 * (observation[POS_X] - coworkerVectors[j][POS_X]) < BLURX ) &&
9          (observation[POS_Y] - coworkerVectors[j][POS_Y]) < BLURY &&
10         (-1 * (observation[POS_Y] - coworkerVectors[j][POS_Y]) < BLURY ) &&
11         observation[FEAT1] == coworkerVectors[j][FEAT1] &&
12         observation[FEAT2] == coworkerVectors[j][FEAT2] &&
13         observation[FEAT3] == coworkerVectors[j][FEAT3]))
14 ...
```

Figure 3: Example of information flow conditions in JML

Fig. 4 shows a slightly simplified specification of the allowed information flow of the method *updateObservation()*.

```
1 public normal_behaviour
2 requires ...
3 respects guestVectors.length, guestVectors,
4     ** list of all features of guests **,
5 containsCoworker(observation),
6 containsGuest(observation),
7     (containsGuest(observation) || !containsCoworker(observation))?
8     (** list of all features in the observation **):
9 (null);
```

Figure 4: Simplified information flow conditions

The *requires* clause in line 2 holds some preconditions that have to be satisfied before the method is called. We skip the details for the sake of brevity. In the *respects* clause, a list of expressions is given, the evaluation of which may be known to the environment before and after the execution of the method call. Line 3 specifies that the amount of guests in the system may be known to the environment.

Line 4 specifies that each feature stored about guests, for example their position, may be known to the environment. The predicate *containsCoworker* (line 6) expresses that there exists an employee object in the system into which the observation (given as a parameter) can be fused. Note that only the existence of an employee may be exposed, no further details about the employee or the amount of employees registered in the system must be exposed. Line 7 specifies that the information whether or not a guest exists into which the observation can be fused may be released. Again only the information about the existence is released, but not the values of the features of the observation or the values of features of the guest object.

Finally, lines 8 and following specify that the values of features passed in observations may only be known to the environment, either if there already exists a guest in the system, which is recognized, or if no employee was recognized. The first case is clear since the features of guests may be known to the environment. The second case specifies that if an observation is made and nothing changes for the operator, obviously an employee was recognized.

We used the KeY Tool for verification of the implementation. It takes Java source code annotated with JML as input and uses symbolic execution in order translate it into Java DL (see details in [5], [6]). The containing sequent calculus is used to prove that the specification is satisfied by the implementation. Details about the implementation of self-composition can be found in [4]. We verified the information flow for the biggest part of our implementation.

CONCLUSION

In this work it is shown that data minimisation is not always the way to maximum privacy protection. In certain cases it is beneficial for privacy to collect more data and use the gained information in an anonymization process. When certain requirements are fulfilled, i.e., the extra data is only used for anonymization purposes and not useable in any other context, it is feasible to achieve better privacy by collecting more data. In particular, we can improve the selectivity of surveillance measures. Referring to our example scenario, we adhere to the privacy requirement of hiding employees by tracking their positions, while at the same time ensuring that positions of employees are never exposed to the environment. Our hitherto results give strong indication that it is feasible to implement and verify a data store with an information fusion algorithm, which ensures that no private data from objects of the protected class, e.g., employees, is ever disposed by the system.

ACKNOWLEDGMENTS

This work was partially funded by the KASTEL project by the Federal Ministry of Education and Research, BMBF 01BY1172, and by Fraunhofer Gesellschaft Internal Programs, Attract 692166. The views expressed are those of the authors alone and not intended to reflect those of the Commission.

REFERENCES

[1] Senior, A., Pankanti, S., Hampapur, A., Brown, L., Tian, Y.-L., Ekin, A., Connell, J., Shu, C. F., and Lu, M. (2005). *Enabling video privacy through computer vision.* Security & Privacy 3(3), pp. 50-57.

[2] Wickramasuriya, J., Datt, M., Mehrotra, S., and Venkatasubramanian, N. (2004). *Privacy protecting data collection in media spaces.* In Proc. 12th annual ACM intl. conf. on Multimedia, pp. 48-55.

[3] Vagts, H., and Beyerer, J. (2011). *Enhancing the acceptance of technology for civil security and surveillance using privacy enhancing technologies.* In Future Security, pp. 372-379.

[4] Scheben, C. and Schmitt, P. H. (2011). Verification of Information Flow Properties of Java Programs without Approximations. FoVeOOS, pp. 232-249.

[5] Beckert B., Hähnle R. and Schmitt P. H. (2007) Verification of Object-Oriented Software: The KeY Approach, LNCS 4334.

[6] Weiß B. (2011) Deductive Verification of Object-Oriented Software: Dynamic Frames, Dynamic Logic and Predicate Abstraction. PhD thesis, Karlsruhe Institute of Technology.

[7] Amtoft T. and Banerjee A. (2004) Information flow analysis in logical form. SAS 2004.

[8] Joshi R. and Leino K. R. M. (2000) A semantic approach to secure information flow. Sci. Comput. Program, pp. 113-138.

[9] Leavens G. T., Baker A. L. and Ruby, C. (2006) Preliminary Design of JML: a behavioral interface specification language for Java. SIGSOFT Software Engineering Notes, pp 1-38.