Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011)

Adaptive Security Architectures for Global Sensing Applications

Daniel Schreckling and Joachim Posegga

10 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Adaptive Security Architectures for Global Sensing Applications

Daniel Schreckling and Joachim Posegga

{ds,jp}@sec.uni-passau.de
IT-Security Group
Institute of IT-Security and Security Law
University of Passau

Abstract: We discuss the problem of providing a security architecture for global sensing applications which are distributed in nature and which allow for a variety of different sensing tasks. We propose to make use of data security policies tagged to data or resources generating, processing, or consuming data on a sensing device. These policies are used to feed an automated information flow analysis process which identifies unsatisfied security constraints. Results of this analysis are used to automatically adjust an application to comply with the given security constraints.

Keywords: global sensing, information flow analysis, inline reference monitors

1 Introduction

If we look at the number of sensor information available in state of the art mobile devices such as laptops, handhelds, or smartphones we see a paradigm change in the sensing domain. Instead of designing small single-purpose hardware platforms which deliver specific sensor information, multi-purpose platforms provide sensor information which can be measured easily and may provide useful input to existing applications. Furthermore, we observe an increasing trend towards peer-to-peer connectivity. It avoids centralised providers or expensive infrastructures and combines, filters, or adapts sensor information to new types of applications.

We consider the following exemplary scenario to illustrate the variety of applications enabled by the increased availability of sensors. Assume a user who carries a mobile device which does not provide sensorial input. Sensor data is consumed as input for the application running on the device. Based on the information the device can collect through its sensors and network interfaces, an application displays the user a route through a known map, e.g. a shopping center. Based on the current location and destination it computes a route depending on the number of people in the building, the user interests or requirements, etc. This route is displayed on a map which is provided by an application on the same device or by the IT-infrastructure of the building.

This *composed application* may be extended further such that a remote route planer is used and computes the route. In this case, the current location and destination become subject to privacy concerns. The interaction between the application hosted on the user's device and the remote route planer requires additional authentication and encryption mechanisms.

To increase the quality of routes the remote planer may additionally ask the users' devices to provide feedback about the environmental status, e.g. humidity, noise, speed of movement, location, etc. Different security mechansims are required depending on how this information is processed, e.g. the server applies anonymization techniques or sends averaged information not

connectable to a user. Instead of using a server the application may also distribute information in a p2p fashion. In the latter case additional anonymisation techniques are required.

When isolating the single components of the application sketched above we observe that each of them can be used separately in different scenarios, serving different tasks. As a consequence, the data processed and delivered to them may induce completely different security requirements. Thus, the security architecture of the application which ensures these requirements must adapt.

However, state of the art security frameworks interrupt the operation of or signal problems with applications which interfere with the security requirements of data. Appropriate configurations to ensure a secure operation of such applications within a specific execution environment are required. Applications are restricted to specific tasks, data, and runtime contexts.

We present the concept of an architecture which aims for the secure operation of these applications without the need for manual reconfiguration. This will allow for flexible combinations of sensing tasks which respect the security requirements of users, devices, and the data associated to or generated by them.

Our paper is organized in six sections. After this introduction Section 2 outlines the general security issues of global sensing applications. Section 3 introduces the envisioned adaptive security architecture. It introduces the application model considered, explains the required types of policies, and outlines the methods for policy propagation and automatic inlining. Before Section 5 concludes our paper, Section 4 lists some of the work that is related to our architecture.

2 Security Issues and Goals

The variety of new applications enabled by global sensing will process sensor or context information. Consequently, data will be condensed, transformed, combined, distributed, new information will be inferred and will be distributed. Inevitably, such processing deviates from the originally intended usage of the data. In order to protect his privacy, the user needs to track the usage of the data he generated. However, such a data centric view yields several problems:

1. Devices will require reconfiguration to meet changing security requirements.
2. Users may not be aware of the fact that they are producing security critical sensor data.
3. The variety of devices, data, applications, and their interactions will overwhelm the user with security conflicts or queries to be solved or answered.
4. Due to modifications and inference of data a user may not understand all security issues.

As the amount of information global sensing scenarios will process is several magnitudes higher than what is experienced in traditional sensor networks, the transparent integration of autonomic security mechanisms is required. *Devices must be able to largely manage themselves* to relieve the user and avoid delegating critical security decisions to non-security experts. Multi-purpose applications must be supported instead of inhibiting their operation: As the precise way data is processed is unknown prior to deployment the *flexibility of security mechanisms* which address data processing issues are vital in global sensing applications.

From the perspective of an application, it is hard to determine the security requirements which ensure an execution without causing conflicts with possible data security policies. Even if there

were a fixed set of sensor inputs, their security requirements are not know in advance. Furthermore, due to system feedback, environmental changes, or user preferences the requirements may change during the lifetime of the system. Therefore, the developer of an application is unable to design an application which complies with all possibly arising security requirements. Thus, a *feasible security architecture must postpone the decision about the deployed security architecture to the time the application is actually used*.

This situation is aggravated by the fact that global sensing applications are intrinsically distributed: They are composed of information providers, such as sensing devices, information processors, such as entities combining or inferring new data, and information consumers, such as other sensing devices. Thus, the application possesses a distributed information flow among various application components with different processing and security properties. As the distribution of components is unknown during application design a *feasible security architecture must be able to dynamically and appropriately adapt*.

Finally, we expect the deployment of global sensing scenarios to inspire the development of high-level, abstract programming concepts. The latter will allow non-expert users to assemble applications satisfying their specific needs. We envision simple application components representing basic functionalities. They can be combined in an intuitive way to construct more complex applications. In this way, users can easily integrate newly derived context information into existing applications. A security architecture will have to *guarantee that such dynamic compositions of applications are not inducing insecure information flows*.

Classic solutions mostly account for rigorous and inflexible solutions based on fixed and centralised security architectures and infrastructures. If new system components are added, new services become available, or if security policies or requirements are changed, the system must be redesigned or at least adapted. Security experts are required for these tasks. Classical scenarios also assume applications to be trusted. Before their deployment they are subject to rigorous testing and manual or semi-automatic analysis. After their integration into the production environment they are considered to be monolithic, fulfilling specific security properties. Applications which do not comply with this strict deployment procedure, which do not respect the security policies of the system, or which simply disobey the general security architecture will fail.

In summary, to support applications emerging from or inspired by the emerging global sensing paradigm, a *security framework is required which will transparently, thus autononmically, integrate into an application* to address the individual and dynamically changing security requirements of a user or the data processed.

3 Adaptive Security Architectures

The architecture we propose is based on the hypothesis that it is impossible to define a security architecture for global sensing scenarios and their applications before their deployment. This is due to the fact that the static components which security mechanisms usually rely upon are reduced to a minimum. The single static characteristic a security mechanism can exploit are the security properties a user can assign to data sinks and sources as well as to data stored on a device. Therefore, we advocate an adaptive security architecture driven by data security policies.

To allow for adaptive security architectures automatic flow analysis is required. It must de-

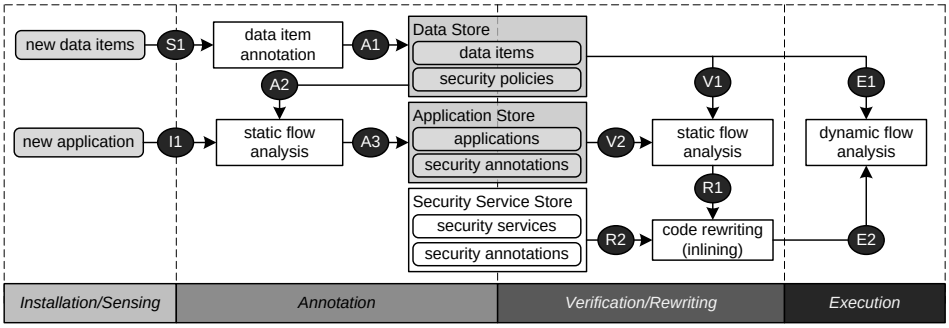

Figure 1: Abstract Operation of Adaptive Security Architectures

termine the type and place at which specific security primitives within a distributed application are required in order to ensure pre-defined security properties (see also Figure 1). Exploiting the insights from the variety of existing data security policies subsumable under the term of "sticky policies" [KSW03] and dynamic labels [ML97, ZM04] we attach security policies to sensors. They specify how data generated or stored on a node, e.g. sensor or application data (S1), must be used (A1). Policies are either defined by users or, more generally, by the party which owns the device. Additionally, all installed applications (I1) are annotated by an automatic information flow analysis process. Without the assistance of a security expert, this process determines which data an application uses in which way and thus which security requirements it has to fulfill (A2). This security annotation is stored with the application (A3). Before an application finally processes data which is tagged with a security policy, another automatic static analysis process propagates policy information along the data flow (V1, V2). This propagation may yield conflicts, i.e. the processing of a data item does not comply with its security policy (R1). In this case, the framework first determines the required security primitives to fulfil the unsatisfied constraints. Afterwards, the architecture of the application is modified and the required security primitives are integrated (R2). In case no security primitive is able to fulfil the security requirements complete control flow branches are truncated. The resulting adapted application (E2) is executed. Security constraints which are only decidable at runtime are enforced dynamically (E1) by reference monitors included in step (R2).

The envisioned system abolishes the need to adapt an application to a given static security architecture. Rather, we automatically integrate required security mechanisms into the application. This helps to design an efficient security framework which deploys security mechanisms only deployed when required and which minimizes user interference. Additionally, we allow for a flexible system that is able to automatically adapt to changing contexts or applications.

3.1 Architecture Model

Our architecture is implemented as a part of the operating system (OS) which runs on a global sensing device. Thus, it can not be influenced directly by an application. In fact, each application runs on top of the framework. In this way we can interfere with the execution of each application at any time. This is comparable with the well-known sandbox execution model. Our prototype,

Constroid, which will integrate the ideas explained below uses the Android platform. This OS already exhibits the features explained above and our security architecture will be placed in the Linux core of the Android system. Accordingly, applications are denoted by Dalvik executables.

An application, which we also call service, consists of one or more services which communicate with each other. Each of these services can run on different nodes. As the application is composed of several services we also call it service composition. A service which can not be split up in other services is called a service cell. This, induces a recursive structure with the service cells at its ends. There are special service cells, called security service cells. They are provided during the deployment of the node, carry specific annotations (see next section) and provide verified security functionalities – security services – when executed.

Finally, applications can access or create data only over a well defined application programming interface provided by our architecture. Any other access is not possible.

3.2 Security Policies

Our architecture is based on data and functional components which are tagged with security relevant information. Thus, security policies take an important role in our work. Due to the space restrictions and the focus of this article we are not going to argue about the human-computer interaction required to specify such policies. We rather argue about the language features required to describe the policies. The language has to be able to

1. describe security requirements a user specifies for a set of data,
2. specify the security state of an application, e.g. whether a user is authenticated
3. describe the influence of a software component on the security state of an application and the data it processes (this may also be the results of an automated analysis of a component),
4. describe "information flow contracts" between distributed software components, i.e. a remote software component delivers a security policy which expresses which operations the component is going to conduct on which data, and
5. support combinations, i.e. policies specified for data should be usable in the context where the same language is used to describe security characteristics of software components.

We distinguish user-, node-, service-, and data policies.

User policies describe the security requirements for a particular user who is using the system. This policy is closely linked to the policies tagged to the data the user owns. A user policy defines the security level the user expects. This includes default settings, i.e. default security policies, for data the user or an application running on behalf of the user generates. Finally, the user policy also describes privacy settings for data which is generated by sensors on the device and which can be linked to the user.

A *node policy* defines the security requirements for a particular device. It can define who is allowed to access the node, under which conditions, which services can be executed by a user, and which resources are accessible. It also specifies security properties for the device, e.g. if the device executes a task without the attendance of a user, a node policy can specify which data it

will provide and which resources can be accessed. Thus, if a device becomes part of a global sensing application the node policy regulates which data is provided and which tasks the node is allowed to perform. Node policies overwrite user policies.

Similarly, a *service policy* is associated with a particular service and defines which security principal can use this service. Furthermore, it describes which security characteristic it has. The latter is generated by a security analysis process which determines the security relevant data flow characteristics and side effects of this service. This information helps the static analysis process to identify unsatisfied security requirements and to retrieve appropriate security services required to secure a service composition. As it will not be possible to completely annotate a service without knowing the security requirements of the data it processes, a *security meta policy format* is introduced. It allows the annotation independent from the actual security requirements by describing flow information and security state changes instead of explicitly specifying them. Finally, service policies are also able to specify contracts between single application components. Due to the distributed nature of global sensing applications this security framework must rely on attestations provided by remote nodes to stick to the specified information and control flow, respectively. Appropriate assurance mechanisms will verify the according compliance.

Finally, the *data security policies* have finest granularity and define the security characteristics for the data items they are assigned to. During the design of an application the developer must generate an ontology for the data the application is going to process. Based on this ontology, the system creates and initiates security policies for single data items defined by ontolgy. The restrictions contained therein can be defined for various subjects, i.e. for users, services, or nodes. Comparable to the service policies, data policies feed the automated static analysis process to verify that services securely process the data they operate on. Additionally, during the execution of a service composition these labels are transformed into and used as dynamic labels.

While we have already designed such a language [Sch09], it is permanently reviewed and adapted to the requirements of the analysis process and to the emerging security framework.

3.3 Information Flow Analysis

The application depicted in Figure 2 consists of service cells a_1 through a_6. Each cell has input ports and output ports represented by numbered rectangles mounted on top and at the bottom respectively. Input ports are connected to output ports, i.e. the service composition requires that results from one service cell become the input for another service cell, e.g. input port 1 of cell a_4 is fed by the output port 2 of cell a_1. The input to the complete service composition is given on the very top. Black solid arrows indicate control-flow constraints which basically determine in which order the cells must be executed. Our automated analysis process first investigates the internal structure of every single service cell, i.e. during the installation of each cell, this structure is analysed. We determine flows, explicit or implicit, between input and output ports and whether a cell accesses some resource in a writing or reading manner, i.e. data sinks and sources are detected. The result of this flow analysis is stored in a service policy as an annotation with the analysed service. We also depict the result of this intra service data flow analysis by dashed-dotted lines within the service cells.

The combination of this information within a specific service composition creates a detailed data flow graph within an application. Thus, we are able to track every data item the service

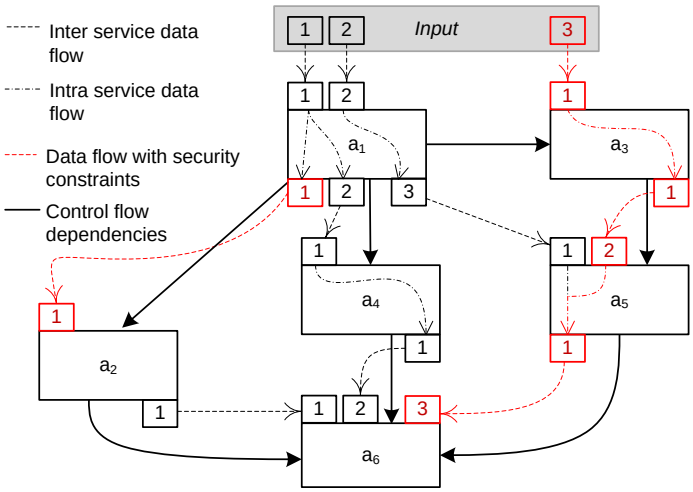

Figure 2: Sample global sensing application with flow annotations

processes in a graph (dashed and dashed dotted lines in Figure 2).

The analysis of the composition starts at the data sources. Security properties are either defined by the specific security policies attached to data or by the respective default policies, e.g. for sensors. They can propagate through the data flow graph towards every possible data sink. During this propagation, the security properties are compared with the service policies of the services processing the data and the default sink policies. If they comply, the data flow has no conflicts and the service composition can be executed. Otherwise, the conflicts have to be resolved or appropriate monitoring mechanisms have to be integrated which is described in Section 3.4.

Assume that input 3 in Figure 2 contains the credit card number of user U, i.e. it is confidential input which should only accessible by U. The red path from input 3 to service cell a_6 denotes the propagation of this security property. Further assume that cell a_6 stores the data from input 3 in a local database which is only accessible by user U and which stores its input in an encrypted fashion. In this case the security properties of the database, the sink, comply with the data item characteristics. However, if cell a_6 was a connection to another node then the service has to ensure that the credit card number is encrypted before it is handed over to the lower layer which would send it to this node.

Figure 2 also shows a security sensitive data flow which ends without reaching a sink. Let service cell a_1 generate data which is subject to security constraints, e.g. by reading a password hash from a source. This data is then provided to another cell, a_2, which simply compares this information with some other data item. Afterwards, the sensitive data is discarded.

3.4 Automatic Inlining of Reference Monitors

The automated analysis process detects paths with conflicting security requirements. It returns all information flow paths with conflicts. Conflicts are described by the specification language sketched in Section 3.2 and report as a list of unsatisfied security constraints. This information guides the automatic reconfiguration process of the service composition: *automatic inlining*.

According to our model from Section 3.1 we can access a set of semantically annotated security service cells. Cells which satisfy the security constraints not met by the composition are candidates for inclusion. An appropriate position for this inclusion requires a good trade-off.

Clearly, it would be straight forward to include, e.g., an authentication mechanism at the top of a service composition. This would imply that the authentication is executed only once and it holds for the whole service, i.e. it does not have to be executed again. However, there may be branches which do not require this security measure. Thus, entire control flow branches of a service may become unusable only because somewhere in the service a credential is required, e.g. for the administrator. The other extreme would be a service which always asks for passwords as soon as it accesses a resource. This would annoy the user and a single authentication dialogue at an appropriate position in the control flow would be more feasible and user friendly. Here, our mechanism also has to ensure that security service cells are not combined arbitrarily to not induce new vulnerabilities through inappropriate combinations. Finally, mechanisms which alter the structure of data, e.g. encryption, and thus prevent the further processing and functionality of the service, must only be included in specific positions of the service composition.

The inclusion of service cells is implemented by using code rewriting. At specific position identified by the analysis process inline reference monitors are included [ES00]. To integrate the reference monitors we deploy two transformation operations: *truncation* and *insertion*. Here, we adopt the terminology and concept of Edit Automata [LBW05] introduced by Ligatti et al.

A *truncation transformation* of information flow paths represents a very restrictive and intrusive operation. It allows the complete removal of paths from an application which must not be adopted due to the security requirements specified for the node. A simple example is a node policy specifying that no location information must be published to a third-party. An application which still publishes this information will infringe this policy and must be rewritten.

In contrast, the *insertion transformation* allows for more fine-grained modifications. This transformation allows the simple insertion of security service cells within the control or information flow, e.g. authentication, encryption, or anonymisation services. Further, the insertion transformation will include instructions which update and check security policies during the information processing. Finally, insertion transformations will also integrate instructions which track global states of the application, e.g. an authenticated session.

Clearly, it is not possible to determine all security constraints and the required enforcement mechanisms during the analysis of a service composition. Thus, every security service cell integrated into the service composition must inspect and update the dynamic labels of data processed by this cell during runtime. So, our inlining process combines static analysis for security with the dynamic inspection of security constraints.

4 Related Work

As our architecture borrows from numerous fields in information security it is impossible to list all related work. Therefore, we mainly focus on relevant contributions in the field of information flow analysis which already offers a a large body of work [SM03].

Recent work developed [CMJL09] efficient implementations of distributed information flow analysis of JavaScript Code. The insights from this work will help to develop appropriate anno-

tation and enforcement mechanisms. However, they appear the expressiveness of this approach is too low to be feasible for our framework.

Enck et al. [EGC+10] and Ontgang et al. [OBM10] also show that the implementation of dynamic enforcement mechanisms on platforms which offer only restricted resources is possible in principle. Our framework can benefit from the results of this work as we must also deploy dynamic analysis methods. However, the static nature of [OBM10] and the purely notifying character of [EGC+10] has to be overcome to be useful in our architecture.

Our architecture will also make use of recent mechanisms which combine labeling and inline referencing [MAS10] and which investigating the power of in-line reference monitors [Erl04, MRS10] as well as Edit Automata [LBW05] and their successors: Mandatory Results Automata [LR10]. However, our work will push the limits of these approaches and of flow analysis and develop mechanisms combining dynamic as well as static flow analysis to dynamically reconfigure distributed systems. Formal frameworks such as MAKS introduced by Mantel [Man03] will provide the required formal basis for this automation.

5 Conclusion

Therefore, we introduce the concept of a security architecture which is able to adapt to the requirements imposed by global sensing applications. Instead of focusing on a merely language or implementation based approach and instead of designing a one-fits-all architecture, our approach aims at the development of an architecture which automatically adapts to the actual security needs. The latter are defined by the user or administrator of a sensing device. They specify the security requirements for every sensor or data item generated. A large set of policies offers an appropriate tool to specifie security constraints for or on users, devices, services, and data. Despite this large set the user mainly interacts with the security system by specifying the security needs for the entities he is familiar with, i.e. the data and the sensors producing data, e.g. sensors. In this way, the user can still understand his influence on the system.

Several aspects of the architecture introduced above have been implemented in a first prototype [CBL+09]. The results obtained from this implementation are promissing and encourage us to promote a sound and holistic definition and implementation of this framework.

Bibliography

[CBL+09] I. Carreras, L. Bassbouss, D. Linner, H. Pfeffer, V. Simon, E. Varga, D. Schreckling, J. Huusko, H. Rivas. BIONETS: Self Evolving Services in Opportunistic Networking Environments. In *Proceedings of BIONETICS 2009*. ICST, December 2009.

[CMJL09] R. Chugh, J. A. Meister, R. Jhala, S. Lerner. Staged information flow for javascript. In *PLDI '09: Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*. Pp. 50–62. ACM, New York, NY, USA, 2009.

[EGC+10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of OSDI 2010*. October 2010.

[Erl04] U. Erlingsson. *The inlined reference monitor approach to security policy enforcement*. PhD thesis, Cornell University, Ithaca, NY, USA, January 2004. Adviser-Schneider, Fred B.

[ES00] U. Erlingsson, F. B. Schneider. IRM Enforcement of Java Stack Inspection. In *In IEEE Symposium on Security and Privacy*. Pp. 246–255. 2000.

[KSW03] G. Karjoth, M. Schunter, M. Waidner. Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies (PET'02)*. Pp. 69–84. Springer-Verlag, Berlin, Heidelberg, 2003.

[LBW05] J. Ligatti, L. Bauer, D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security* 4:2–16, 2005.

[LR10] J. Ligatti, S. Reddy. A Theory of Runtime Enforcement, with Results. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. Sept. 2010.

[Man03] H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, Juli 2003.

[MAS10] J. Magazinius, A. Askarov, A. Sabelfeld. A lattice-based approach to mashup security. In *ASIACCS '10: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. Pp. 15–23. ACM, New York, NY, USA, 2010.

[ML97] A. C. Myers, B. Liskov. A Decentralized Model for Information Flow Control. In *In Proc. 17th ACM Symp. on Operating System Principles (SOSP*. Pp. 129–142. 1997.

[MRS10] J. Magazinius, R. Russo, A. Sabelfeld. On-the-fly inlining of dynamic security monitors. In *In Proc. IFIP International Information Security Conference*. 2010.

[OBM10] M. Ongtang, K. Butler, P. McDaniel. Porscha: Policy Oriented Secure Content Handling in Android. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC 2010)*. Austin, TX, USA, August 2010.

[Sch09] Schreckling, Daniel. Adaptive Security in BIONETS. BIONETS (IST-2004-2.3.4 FP6-027748), Deliverable D4.4, Feburary 2009.

[SM03] A. Sabelfeld, A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* 21(1):5–19, January 2003.

[ZM04] L. Zheng, A. C. Myers. Dynamic security labels and noninterference. In *In Proc. 2nd Workshop on Formal Aspects in Security and Trust, IFIP TC1 WG1.7*. Pp. 27–40. Springer, 2004.