

# Towards Information Flow Auditing in Workflows

Claus Wonnemann  
Department of Telematics  
Albert-Ludwigs-Universität Freiburg, Germany  
wonnemann@iig.uni-freiburg.de

## Abstract:

The paper proposes an approach for compliance audits in workflow environments based on the tracking of information flow. Requirements are formalized as a binary relation on the workflow principals. The workflows' execution logs are transferred into graph-based representations of the explicit information flows (dataflows) and adherence to compliance requirements is checked while traversing these graphs. The scope and limits are discussed and the major milestones for further work are outlined.

## 1 Introduction

Many compliance requirements that are applicable to IT systems somehow regulate the flow of information, e.g. when some sensitive information must not be disclosed to unauthorized parties, or if some datum has to pass a number of checks before being released (e.g. to ensure the four-eyes-principle). Such information flow constraints can be expressed as relations among system principals, independently from the concrete implementation. This formalization is stronger than access control policies because it captures information propagation throughout the system (*end-to-end*) rather than access at certain points. Further, it captures *implicit* information flows, which lie outside the scope of access control mechanisms (see below).

This paper proposes an approach for compliance audits in workflow environments based on the a-posteriori tracking of information flow. We present the IFAudit system, which consists of a policy language for the expression of information flow constraints and an audit algorithm to track explicit information flow (dataflow), based on log data. We discuss the scope and limits of the implementation and outline the most important milestones for further work.

**Types of Information Flow** There are two types of information flow: explicit and implicit. Explicit flow is information transfer through “legitimate” channels that are intended for this purpose, such as messages and shared storage. Implicit flow describes the extraction of information through *inference*. This is the case when multiple data items are combined to yield information, e.g. when anonymized patient records are matched against address directories to reduce the anonymity set and, eventually, reveal a patient's identity. Information can also be inferred from the observable behavior of the system. For instance,

by observing the response times of a crypto system, an attacker might infer knowledge on its private key. Here, the system’s timing behavior forms a *covert channel* that enables an attacker to obtain secret information.

## 2 Workflows and Log Data

Workflows formally describe business processes as structured sequences of activities. The execution of a workflow is coordinated by an execution engine which triggers activities, synchronizes their interaction and communication, and writes log data. These log files are the basis for the audits considered in this paper.

Log files of workflow systems are structured as sequences of entries that each represent a workflow activity. An entry comprises (among other attributes) the originator, on which behalf the activity was performed, and the data items that were read (input) and written (output) by the activity. Further, it is assumed that each entry can be attributed to a workflow instance and that the entries are totally ordered (e.g. by providing a timestamp).

Expressed formally,  $\mathcal{L}_{\mathcal{W}}$  denotes the log file for a workflow  $\mathcal{W}$ , with  $\mathcal{L}_{\mathcal{W}} \in \mathcal{A}^*$ , where  $\mathcal{A}$  is the universe of all log file entries.  $\mathcal{L}_{\mathcal{W}}$  can be partitioned into a set  $\{i_0^{\mathcal{W}}, i_1^{\mathcal{W}}, \dots, i_n^{\mathcal{W}}\}$  where  $i_i^{\mathcal{W}}$  denotes the trace (ordered sequence of log entries) for the  $i^{\text{th}}$  execution of  $\mathcal{W}$  ( $0 \leq i \leq n$ ). An entry  $a$  has the attributes *orig* (the originator), *input* (the set of input data items) and *output* (the set of output data items). The attributes of an entry are referenced with a dot (e.g.  $a.\text{orig}$ ). For convenience, *activity* and *log entry* (of the activity) are used interchangeably in the following.

## 3 Policies

An information flow policy  $\mathcal{P} = \{r_1, \dots, r_n\}$  is a set of rules that specify which principals in a system may or must not exchange information. The principals in our setting are the originators (subjects) that appear in a log file  $\mathcal{L}_{\mathcal{W}}$ , denoted as  $\mathcal{S}_{\mathcal{W}} = \bigcup_{a \in \mathcal{L}_{\mathcal{W}}} a.\text{orig}$ .

The function  $sc : \mathcal{S} \rightarrow \mathcal{D}$  assigns principals security domains from the set  $\mathcal{D}$  of domains. The information flow policy specifies among which security domains information may or must not be exchanged. An activity has the same domain as its originator.

Fig. 1(a) shows a typical multi-level security policy with three domains (a crossed arrow indicates that there must not be information flow). Compliance requirements often demand that some data has to take a specific path, for instance, to ensure that financial statements are checked by the revision department before being published to investors. Fig. 1(b) depicts a corresponding policy: information coming from the *Accounting* domain must not flow directly to the *IR* department, but has to pass through *Revision*.

A policy rule has the form *Restriction*  $\Rightarrow$  *Exception*. *Restriction* is a flow relation  $source \rightsquigarrow target$ , which specifies that information must not flow from domain *source* to domain *target* ( $source, target \in \mathcal{D}$ ). *Exception* is a logical combination of flow re-

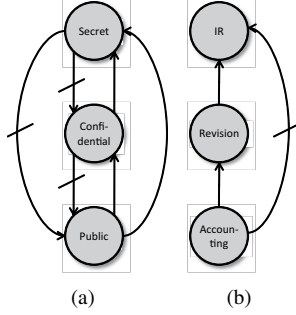


Figure 1: Example Flow Policies.

```

<Policy>      ::= <Rule> | <Rule>, <Policy>
<Rule>       ::= <Restriction> =>
                <Exception>
<Restriction> ::= true | <FlowRel>
<Exception>  ::= false | <FR-DNF>
<FR-DNF>    ::= (<ConClause>) |
                (<ConClause>) ∨ <FR-DNF>
<ConClause>  ::= <FlowRel> |
                <FlowRel> ∧ <ConClause>
<FlowRel>   ::= <Domain>↔<Domain>
<Domain>    ::=  $d \in \mathcal{D}$ 

```

Figure 2: Policy grammar in BNF.

lations in disjunctive normal form which defines legitimate flows that might contradict *Restriction* (as in the example in Fig. 1(b)). Fig. 2 gives the grammar for these policies in Backus-Naur-Form.

Using this grammar, the example policies from Fig. 1 have the form

```

Secret↔Confidential => false,
Confidential↔Public => false,
Secret↔Public => false

```

and

```

Accounting↔IR => (Accounting↔Revision ∧ Revision↔IR),

```

respectively.

Besides the (extensional) specification of flow relations, which express whether information may flow between domains, an information flow policy must define which (patterns of) system actions constitute information transmission. In its current state, the IFAudit system only supports the detection of dataflow, i.e. explicit information flow: There is an information flow from domain  $d_1$  to domain  $d_2$ , if (and only if) there is a data item which has been modified by an activity from  $d_1$  and is subsequently read by an activity from  $d_2$ . This approach is similar to the tainting mode in the Perl programming language. It is sufficiently expressive to capture common compliance violations, such as failure to “sanitize” information or illicit propagation of some data item. A more comprehensive and fine-grained analysis is subject of ongoing work (see Section 6).

## 4 Audit Algorithm

The IFAudit system analyzes explicit information flow on the workflow level, i.e. the information transfer that occurs among the activities. Activities are regarded as black boxes of which the internal workings are not necessarily known, but only their I/O. On this abstraction level, IFAudit allows to check whether a workflow instance has exhibited illicit

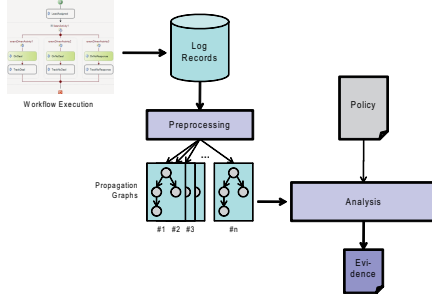


Figure 3: Outline of the IFAudit system.

```

VISIT( $u$ ):
for each (Rule  $r \in OpenRules$  that is closed by  $u$ ) do
  Report violation of  $r$  through flow(s) from  $SRC(r)$  to  $u$ ;
for each (Rule  $r \in \mathcal{P}$  that is opened by  $u$ ) do
   $OpenRules := OpenRules \cup r$ ;
   $SRC(r) = SRC(r) \cup u$ ;
Mark  $u$  as visited;
 $Successors :=$  Direct successors of  $u$ ;
Sort  $Successors$  in ascending order;
for each ( $v$  in  $Successors$ ) do
  if ( $v$  is not marked as visited) then
    VISIT( $v$ );
for each (Rule  $r \in \mathcal{P}$  that is opened by  $u$ ) do
   $SRC(r) = SRC(r) \setminus u$ ;
  if ( $SRC(r) = \{\}$ ) then
     $OpenRules := OpenRules \setminus r$ ;

```

Figure 4: Pseudo-code for VISIT.

information flows with respect to a specified policy.

The structure of IFAudit is outlined in Fig. 3. The log entries of each workflow instance are transferred into a *propagation graph* (PG) that represents the explicit information flows (dataflows) between the involved principals. The PGs are subsequently analyzed for illicit information flows according to the given policy. The PG for a trace  $i_n^W \subseteq \mathcal{L}_W$  is denoted  $PG_n^W$ . It is defined as a graph  $(V, E)$ , where  $V = \{a \in \mathcal{A} \mid a \in i_n^W\}$  is a set of activities (representing the graph’s vertices), and  $E = \{(a, b) \in (\mathcal{A} \times \mathcal{A}) \mid a < b \wedge a.output \cap b.input \neq \{\}\}$  is an asymmetric relation representing the graph’s (directed) edges. An edge  $(a, b)$  in a PG indicates that there was a dataflow from activity  $a$  to activity  $b$ .

**Policy Check** To check whether a workflow instance adheres to a given policy, its PG is traversed in a depth-first fashion. During traversal, it is checked whether an activity marks the starting point of an illicit information flow (i.e. if it is assigned a security domain that is the source of a rule’s restriction). Such rules are kept in a set of “open” rules and checked whether they are “closed” (i.e. violated) by a subsequent activity. An activity closes a rule, if its assigned security class corresponds to the end point of that rule’s restriction, and if the rule’s *exception*-clause evaluates to *false*. Fig. 4 shows the pseudo-code for the procedure VISIT, which checks whether an activity closes or opens a rule before recursively applying itself to each of the activity’s successors. The field *OpenRules* contains the set of rules that have been opened by one or many activities residing above the current activity in the PG. The mapping *SRC* maps each rule from *OpenRules* to the set of activities that have opened that rule.

## 5 Related Work

The Examina system allows the formalization and a-posteriori checking of privacy policies [Acc08]. However, there is no explicit notion of information flow, i.e. information cannot be tracked over several data items. Research in *process mining* focuses on building models of the process’ control structure (e.g. using Petri nets) [vdAWM04]; concerning

security aspects, there is some work addressing the detection of anomalous process behavior [vdAdM05]. Research on information flow control concentrates almost exclusively on the prevention of illicit flow through static and dynamic methods [SM03]. The a-posteriori detection has previously been addressed by the author [AW09].

## 6 Conclusion and Future Work

The IFAudit system allows to check whether information in workflow applications has flown along specified paths. Therefore it can validate adherence to those compliance rules that constrain information propagation. In its current state, IFAudit is restricted to dataflow, which is arguably the most relevant type of information flow in the compliance context. Since information flow in a computer system is, in general, undecidable [DD77], IFAudit uses a common approximation: it is assumed that an activity reading a datum completely absorbs the contained information and passes it entirely to every output datum. This is a conservative assumption which ensures that every (explicit) information flow is captured. The drawback is that the system is likely to generate many false positives, when an informational dependency is assumed where there is not one. Therefore, precision improvement is a major subject of current and future work (see below).

We currently evaluate the correctness and efficiency of the IFAudit system with a proof-of-concept implementation. For this purpose, a simulation environment for workflows was built, which can execute multiple instances of a workflow model (parametrized with varying input) and write the corresponding log files. Further, it is planned to extend IFAudit in multiple directions:

**Inference** The accumulation of multiple data items causes illicit flow, if, for instance, multiple anonymized patient records can be combined to reveal an identity. In order to detect such flows, we plan to integrate a mechanism that allows the formalization of inference capabilities in terms of inference rules. This way, it can be checked whether there might occur illicit flow whenever multiple data items converge at some principal, possibly in the course of multiple executions. Inference rules can also be used to formalize common compliance policies such as Chinese Wall: here it must be ruled out that information from competing clients ever converge at the same principal.

**Formalization and Detection of Implicit Flow** To capture implicit information flows through covert channels, more sophisticated flow semantics than tainting are needed. The classical notion is *noninterference*, the requirement that the interactions of higher-level users with a system must not influence the interactions of lower-level users [GM82].

We are currently working on an audit algorithm for the detection of implicit flow based on information theory. It measures how much the entropy of hidden (high-level) variables decreases when public (low-level) variables can be observed. If the entropy drops significantly, there is information transmission from the hidden to the public variable. Conversely, noninterference holds when the entropy does not change at all. The advantage of

the audit approach compared to static methods is that the transferred information can be precisely quantified (with respect to the given log data). While static methods can determine whether the possibility of implicit information transfer exists (in principle), audits can determine whether such a transfer has indeed occurred, and to which extent.

**Declassification** Generic flow semantics such as noninterference in its original form are too restrictive for most applications, since they characterize information flows through causal flows and reject secure programs where there is a causal flow without an associated information flow [Rya01]. Further, certain “downward” flows are necessary in virtually every application: a standard password check is considered insecure because even the rejection of a wrong guess reveals some information on the password (namely, what it is *not*). Instead of requiring no inference between high and low *at all*, it is more reasonable to require that the higher-level behavior must not influence lower-level observations *in a critical way*. This issue is known as *declassification* (for an overview see [SS05]). We are currently seeking to integrate a suitable downgrading mechanism into IFAudit that allows to consider encrypted channels and other “sanitizing” effects in a workflow.

## References

- [Acc08] Rafael Accorsi. Automated Privacy Audits to Complement the Notion of Control for Identity Management. In *Policies and Research in Identity Management*. Springer, 2008.
- [AW09] Rafael Accorsi and Claus Wonnemann. Detective Information Flow Analysis for Business Processes. In *BPSC*, pages 223–224, 2009.
- [CS08] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *Proceedings 21st IEEE CSF*, pages 51–65, Washington, DC, USA, 2008. IEEE Computer Society.
- [DD77] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, 1977.
- [GM82] Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *Proceedings IEEE Symposium on Security and Privacy*, pages 11–20, 1982.
- [Rya01] Peter Y. A. Ryan. Mathematical Models of Computer Security. In *Foundations of Security Analysis and Design*, pages 1–62, London, UK, 2001. Springer-Verlag.
- [SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
- [SS05] Andrei Sabelfeld and David Sands. Dimensions and Principles of Declassification. In *Proceedings 18th IEEE CSFW*, pages 255–269. IEEE Computer Society, 2005.
- [vdAdM05] W. v.d.Aalst and A. de Medeiros. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. *ENTCS*, 121:3 – 21, 2005.
- [vdAWM04] W. v.d.Aalst, T. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *TKDE*, 16(9):1128–1142, 2004.