Preventing Unauthorized Data Flows

Emre Uzun[1(✉)], Gennaro Parlato[2], Vijayalakshmi Atluri[3], Anna Lisa Ferrara[2], Jaideep Vaidya[3], Shamik Sural[4], and David Lorenzi[3]

[1] Bilkent University, Ankara, Turkey
emreu@bilkent.edu.tr
[2] University of Southampton, Southampton, UK
gennaro@ecs.soton.ac.uk, al.ferrara@soton.ac.uk
[3] MSIS Department, Rutgers Business School, Newark, USA
{atluri,jsvaidya,dlorenzi}@cimic.rutgers.edu
[4] Department of Computer Science and Engineering,
IIT Kharagpur, Kharagpur, India
shamik@cse.iitkgp.ernet.in

Abstract. Trojan Horse attacks can lead to unauthorized data flows and can cause either a confidentiality violation or an integrity violation. Existing solutions to address this problem employ analysis techniques that keep track of all subject accesses to objects, and hence can be expensive. In this paper we show that for an unauthorized flow to exist in an access control matrix, a flow of length one must exist. Thus, to eliminate unauthorized flows, it is sufficient to remove all one-step flows, thereby avoiding the need for expensive transitive closure computations. This new insight allows us to develop an efficient methodology to identify and prevent all unauthorized flows leading to confidentiality and integrity violations. We develop separate solutions for two different environments that occur in real life, and experimentally validate the efficiency and restrictiveness of the proposed approaches using real data sets.

1 Introduction

It is well known that access control models such as Discretionary Access Control (DAC) and Role Based Access Control (RBAC) suffer from a fundamental weakness – their inability to prevent leakage of data to unauthorized users through malware, or malicious or complacent user actions. This problem, also known as a Trojan Horse attack, may lead to an unauthorized data flow that may cause either a confidentiality or an integrity violation. More specifically, (i) a *confidentiality violating flow* is the potential flow of sensitive information from trusted users to untrusted users that occurs via an illegal read operation, and

The work of Parlato and Ferrara is partially supported by EPSRC grant no. EP/P022413/1.

G. Livraga and S. Zhu (Eds.): DBSec 2017, LNCS 10359, pp. 41–62, 2017.
DOI: 10.1007/978-3-319-61176-1_3

(ii) *integrity violating flow* is the potential contamination of a sensitive object that occurs via an illegal write operation by an untrusted user. We now give an example to illustrate these two cases.

Table 1. Access control matrix

Subject	o_1	o_2	o_3	o_4	o_5	o_6	o_7
s_1	r	r	w	w	w		
s_2	r	r	w	w	w		
s_3			r	r	r	w	w
s_4			r	r	r	w	w
s_5						r	

Example 1. Consider a DAC policy represented as an access control matrix given in Table 1 (r represents *read*, and w represents *write*).

Confidentiality Violating Flow: Suppose s_3 wants to access data in o_1. s_3 can simply accomplish this (without altering the access control rights) by exploiting s_1's read access to o_1. s_3 prepares a malicious program disguised in an application (i.e., a *Trojan Horse*) to accomplish this. When run by s_1, and hence using her credentials, the program will read contents of o_1 and write them to o_3, which s_3 can read. All this is done without the knowledge of s_1. This unauthorized data flow allows s_3 to *read* the contents of o_1, without explicitly accessing o_1.

Integrity Violating Flow: Suppose s_1 wants to contaminate the contents of o_6, but she does not have an explicit write access to it. She prepares a malicious program. When this is run by s_3, it will read from o_3, that s_1 has write access to and s_3 has read access, and write to o_6 using s_3's credentials, causing o_6 to be contaminated by whatever s_1 writes to o_3. This unauthorized flow allows s_1 to *write* to o_6 without explicitly accessing o_6.

Such illegal flows can occur in the many of the most common systems that we use today because they employ DAC policies instead of a more restrictive MAC policy [2]. For example, in UNIX, the key system files are only readable by root, however, the access control rights of the other files are determined solely by the users. If a Trojan horse program is run with the root user's privileges, the data in the system files, such as the user account name and password hashes could be leaked to some untrusted users. As another example, a similar flow might occur in Social Networks as well. For instance, Facebook offers a very extensive and fine-grained privacy policy to protect the data posted on user profiles. However, this policy is under the user's control. A Trojan horse attack is likely when the users grant access to third party Facebook applications, that usually request

access to user profile data. An untrusted application could violate the user's privacy settings and access confidential information.

The first step for eliminating occurrences like the ones depicted in the example above is to perform a security analysis. To date, existing solutions to address such problems give the impression that such unauthorized flows could only be efficiently prevented in a dynamic setting (i.e., only by examining the actual operations), while preventing them in a static setting (i.e., by examining the authorization specifications) would require the computation of the transitive closure and therefore be very expensive. However, in this paper, we show that a transitive closure is not needed for the static case and less expensive analyses can be used to solve this problem. More precisely, we have discovered that merely identifying and then restricting a single step data flow, as opposed to the entire path, is sufficient to prevent the unauthorized flow. This new insight has significantly changed the dimensions of the problem and allows us to offer a variety of strategies that fit different situational needs.

Consider the following situations which have differing solution requirements. For example, in embedded system environments complex monitors cannot be deployed due to their computation or power requirements and therefore existing dynamic preventive strategies are not applicable. Similarly, there are solutions for cryptographic access control [8,13], where accesses are not mediated by a centralized monitor and therefore easily offer distributed trust. In such cases, the access control policy needs to be "data leakage free" by design. In other situations, when there are no special computational or power constraints, a monitor can be used, and therefore can be utilized to prevent data leakages. However, there may also be situations where access needs to be granted even if a data leakage may occur and then audited after the fact. This would happen in emergencies, which is why break-glass models exist [5,23,25].

Therefore, in this paper, we develop different solutions to address both the confidentiality and integrity violations. Specifically, we propose a data leakage free by design approach that analyzes the access control matrix to identify "potential" unauthorized flows and eliminates them by revoking necessary read and write permissions. Since this eliminates all potential unauthorized flows, regardless of whether they actually occur or not, this could be considered too restrictive. However, it is perfectly secure in the sense that no data leakages can ever occur, and of course this is the only choice when monitoring is not feasible. Although it may seem very restrictive in the first place, we apply this only to the *untrusted* sections of the access control system. It is important to note that in all potential unauthorized flows one can only be sure of a violation by performing a content analysis of the objects. This is outside the scope of the paper. We also develop a monitor based approach, in which object accesses are tracked dynamically at each read and write operation. Thus, any suspicious activity that could lead to an unauthorized data flow can be identified and prevented at the point of time that it occurs. Thus, this approach only restricts access if there is a *signal* for an unauthorized data flow.

The fact that it is adequate to identify and eliminate one-step flows allows us to identify a limited set of accesses that are both necessary and sufficient to prevent all confidentiality and integrity violations. On the other hand, earlier approaches proposed in the literature [17,21,32] keep track of all the actions and maintain information relevant to these to eliminate unauthorized flows, and therefore are more expensive than our proposed approach. Moreover, while Mao et al. [21] and Zimmerman et al. [32] address the issue of integrity violation, Jaume et al. [17] address the issue of confidentiality violation, however, none of them tackle both of these problems.

This paper is organized as follows. In Sect. 2, we present preliminary background for our analysis, and in Sects. 3 and 4 we present the details of the two strategies. In Sect. 5, we present the results of our empirical evaluation. In Sect. 6, we review the related work. In Sect. 7, we give our concluding remarks and provide an insight into our future work on this problem. Some of the proofs of the theorems and lemmas are presented in the Appendix.

2 Preliminaries

Access Control Systems. An *access control system* (ACS for short) $\mathcal{C}$ is a tuple $(S, O, \rightarrow_r, \rightarrow_w)$, where S is a finite set of *subjects*, O is a finite set of *objects*, $\rightarrow_r \subseteq O \times S$, and $\rightarrow_w \subseteq S \times O$. We always assume that O and S are disjoint. A pair $(o, s) \in \rightarrow_r$, also denoted $o \rightarrow_r s$, is a permission representing that subject s can *read* object o. Similarly, a pair $(s, o) \in \rightarrow_w$, denoted $s \rightarrow_w o$, is a permission representing that subject s can *write* into object o. For the sake of simplicity, we consider only read and write permissions as any other operation can be rewritten as a sequence of read and write operations.

Graph Representation of ACS. An ACS can be naturally represented with a bipartite directed graph [30]. The *graph of an ACS*, $\mathcal{C} = (S, O, \rightarrow_r, \rightarrow_w)$, denoted $G_{\mathcal{C}}$, is the bipartite graph $(S, O, \rightarrow)$ whose partition has the parts S and O with edges $\rightarrow = (\rightarrow_r \cup \rightarrow_w)$. Figure 1 shows the graph representation of the ACS shown in Table 1.

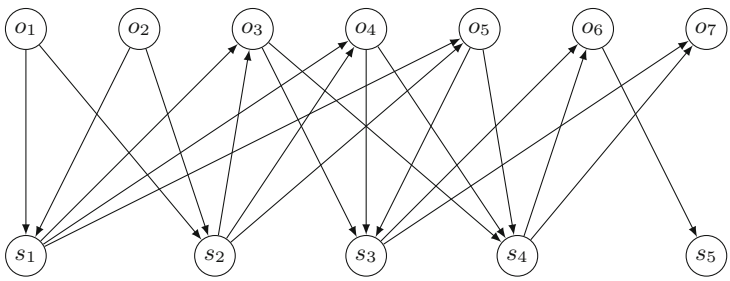

Fig. 1. Graph representation of the ACS given in Table 1

Vulnerability Paths. In an access control system $\mathcal{C}$, a *flow path* from object o to object o', denoted $o \rightsquigarrow o'$, is a path in $G_{\mathcal{C}}$ from o to o', which points out the possibility of copying the content of o into o'. The *length* of a flow path corresponds to the number of subjects along the path. For example, $o_1 \rightarrow_r s_1 \rightarrow_w o_3$ (denoted as $o_1 \rightsquigarrow o_3$) is a flow path of length 1, while $o_1 \rightarrow_r s_1 \rightarrow_w o_3 \rightarrow_r s_3 \rightarrow_w o_6$ (denoted as $o_1 \rightsquigarrow o_6$) is a flow path of length 2 of the ACS shown in Fig. 1. In all, there are 12 flow paths of length 1, while there are 4 flow paths of length 2 in the ACS shown in Fig. 1.

Confidentiality Vulnerability: An ACS $\mathcal{C}$ has a *confidentiality vulnerability*, if there are two objects o and o', and a subject s such that $o \rightsquigarrow o' \rightarrow_r s$ (*confidentiality vulnerability path* or simply *vulnerability path*), and $o \not\rightarrow_r s$. A confidentiality vulnerability, shows that subject s (the *violator*) can *potentially* read the content of object o through o', though s is not allowed to read directly from o. We represent confidentiality vulnerabilities using triples of the form (o, o', s). For example, the ACS depicted in Fig. 1 has the confidentiality vulnerability (o_1, o_3, s_3) since $o_1 \rightsquigarrow o_3$ and $o_3 \rightarrow_r s_3$ but $o_1 \not\rightarrow_r s_3$. Similarly, (o_2, o_6, s_5) is another confidentiality vulnerability since $o_2 \rightsquigarrow o_6$ and $o_6 \rightarrow_r s_5$ but $o_2 \not\rightarrow_r s_5$. In total, there are 15 confidentiality vulnerabilities:
(o_1, o_3, s_3), (o_1, o_3, s_4), (o_1, o_4, s_3), (o_1, o_4, s_4), (o_1, o_5, s_3), (o_1, o_5, s_4), (o_2, o_3, s_3), (o_2, o_3, s_4), (o_2, o_4, s_3), (o_2, o_4, s_4), (o_2, o_5, s_3), (o_2, o_5, s_4), (o_5, o_6, s_5), (o_1, o_6, s_5), (o_2, o_6, s_5).

Integrity Vulnerability: An ACS $\mathcal{C}$ has an *integrity vulnerability*, if there exist a subject s, and two objects o and o' such that $s \rightarrow_w o$, $o \rightsquigarrow o'$ (*integrity vulnerability path* or simply *vulnerability path*) and $s \not\rightarrow_w o'$. An integrity vulnerability, shows that subject s (the *violator*) can indirectly write into o' using the path flow from o to o', though s is not allowed to write directly into o'. We represent integrity vulnerabilities using triples of the form (s, o, o'). For example, the ACS depicted in Fig. 1 has the integrity vulnerability (s_1, o_3, o_6) since $o_3 \rightsquigarrow o_6$ and $s_1 \rightarrow_w o_3$ but $s_1 \not\rightarrow_w o_6$. In total, there are 12 integrity vulnerabilities:
(s_1, o_3, o_6), (s_1, o_3, o_7), (s_1, o_4, o_6), (s_1, o_4, o_7), (s_1, o_5, o_6), (s_1, o_5, o_7), (s_2, o_3, o_6), (s_2, o_3, o_7), (s_2, o_4, o_6), (s_2, o_4, o_7), (s_2, o_5, o_6), (s_2, o_5, o_7).

When an ACS has either a confidentiality or an integrity vulnerability, we simply say that $\mathcal{C}$ has a *vulnerability*, whose *length* is that of its underlying vulnerability path. Thus, for the ACS depicted in Fig. 1, there are $15 + 12 = 27$ vulnerabilities.

Data Leakages. A vulnerability in an access control system does not necessarily imply that a data leakage (confidentiality or integrity violation) occurs. Rather, a leakage can potentially happen unless it is detected and blocked beforehand, using for example a monitor. Before we define this notion formally, we first develop the necessary formalism.

A *run* of an ACS $\mathcal{C}$ is any finite sequence $\pi = (s_1, op_1, o_1) \ldots (s_n, op_n, o_n)$ of triples (or *actions*) from the set $S \times \{read, write\} \times O$ such that for every $i \in [1, n]$ one of the following two cases holds:

(Read) $op_i = read$, and $o_i \rightarrow_r s_i$;
(Write) $op_i = write$, and $s_i \rightarrow_w o_i$.

A run π represents a sequence of allowed read and write operations executed by subjects on objects. More specifically, at step $i \in [n]$ subject s_i accomplishes the operation op_i on object o_i. Furthermore, s_i has the right to access o_i in the op_i mode. A run π has a *flow* from an object $\hat{o}_1$ to a subject $\hat{s}_k$ provided there is a flow path $\hat{o}_1 \rightarrow_r \hat{s}_1 \rightarrow_w \hat{o}_2 \ldots \hat{o}_k$ and $\hat{o}_k \rightarrow_r \hat{s}_k$ such that $(\hat{s}_1, read, \hat{o}_1)(\hat{s}_1, write, \hat{o}_2) \ldots (\hat{s}_k, read, \hat{o}_k)$ is a sub-sequence of π. Similarly, we can define flows from subjects to objects, objects to objects, and subjects to subjects.

Confidentiality Violation: A run π of an ACS $\mathcal{C}$ has a *confidentiality violation*, provided there is a confidentiality vulnerability path from an object o to a subject s and π has a flow from o to s. An ACS $\mathcal{C}$ has a *confidentiality violation* if there is a run of $\mathcal{C}$ with a confidentiality violation.

Thus, for example, in the ACS depicted in Fig. 1, a *confidentiality violation* would occur if there was a sequence $(s_1, read, o_1)(s_1, write, o_3)(s_3, read, o_3)$ which was a sub-sequence of π.

Integrity Violation: A run π of an ACS $\mathcal{C}$ has an *integrity violation*, provided there is an integrity vulnerability path from a subject s to an object o and π has a flow from s to o. An ACS $\mathcal{C}$ has an *integrity violation* if there is a run of $\mathcal{C}$ with an integrity violation.

As above, in the ACS depicted in Fig. 1, a *integrity violation* would occur, for example, if there was a sequence $(s_2, write, o_4)(s_3, read, o_4)(s_3, write, o_7)$ which was a sub-sequence of π.

An ACS has a *data leakage* if it has either a confidentiality or an integrity violation. From the definitions above it is straightforward to see that the following property holds.

Proposition 1. *An access control system is data leakage free if and only if it is vulnerability free.*

The direct consequence of the proposition above suggests that a vulnerability free access control system is data leakage free by design, hence it does not require a monitor to prevent data leakages.

Fundamental Theorem. We now prove a simple and fundamental property of ACS that constitutes one of the building blocks for our approaches for checking and eliminating vulnerabilities/data leakages as shown later in the paper.

Theorem 1. *Let $\mathcal{C}$ be an access control system. $\mathcal{C}$ has a vulnerability only if $\mathcal{C}$ has a vulnerability of length one. In particular, let $\rho = o_0 \rightarrow_r s_0 \rightarrow_w o_1 \ldots s_{n-1} \rightarrow_w o_n$ be vulnerability path of minimal length. Then, if ρ is a confidentiality*

(resp., integrity) vulnerability then $o_0 \rightarrow_r s_0 \rightarrow_w o_1$ *(resp.,* $o_0 \rightarrow_r s_0 \rightarrow_w o_n$*) is a confidentiality (resp., integrity) vulnerability of length one.*

Proof. The proof is by contradiction. Assume that n is greater than one by hypothesis. We first consider the case of confidentiality vulnerability. Let s be the violator. Since ρ is of minimal length, all objects along ρ except o_0 can be directly read by s (i.e., $o_i \rightarrow_r s$ for every $i \in [1, n]$), otherwise there is an confidentiality vulnerability of smaller length. Thus, $o_0 \rightarrow_r s_0 \rightarrow_w o_1$ is a confidentiality vulnerability of length one, as s can read from o_1 but cannot read from o_0. A contradiction.

We give a similar proof for integrity vulnerabilities. Again, since ρ is of minimal length, all objects along ρ, except o_0, can be directly written by s_0, i.e., $s_0 \rightarrow_w o_i$ for every $i \in \{1, \ldots, n\}$. But, this entails that $o_0 \rightarrow_r s_0 \rightarrow_w o_n$ is an integrity vulnerability of length one (as s can write into o_0 but cannot directly write into o_n). Again, a contradiction.

We now present two alternative strategies for preventing data flows, which fit different environments.

3 Access Control Systems Data Leakage Free by Design

When a monitor is not possible or even doable the only solution to get an access control that is free of data leakages is that of having the ACS free of vulnerabilities (see Proposition 1). In this section, we propose an automatic approach that turns any ACS into one free of vulnerabilities by revoking certain rights.

This can be naively achieved by removing all read and write permissions. However, this would make the whole approach useless. Instead, it is desirable to minimize the changes to the original access control matrix so as not to disturb the users' ability to perform their job functions, unless it is absolutely needed. Furthermore, the removal of these permissions should take into account the fact that some of them may belong to *trusted users* (i.e. subjects), such as system administrators, and therefore we want to prevent the removal of these permissions.

We show that this problem is NP-complete (see Sect. 3.1). Therefore, an efficient solution is unlikely to exist (unless P = NP). To circumvent this computational difficulty, we propose compact encodings of this optimization problem into integer linear programming (ILP) by exploiting Theorem 1 (see Sects. 3.2 and 3.3). The main goal is that of leveraging efficient solvers for ILP, which nowadays exist. We show that this approach is promising in practice in Sect. 5.

Maximal Data Flow Problem (MDFP). Let $\mathcal{C} = (S, O, \rightarrow_r, \rightarrow_w)$ be an access control system, and $T = (\rightarrow_r^t, \rightarrow_w^t)$ be the sets of *trusted permissions* where $\rightarrow_r^t \subseteq \rightarrow_r$ and $\rightarrow_w^t \subseteq \rightarrow_w$. A pair $Sol = (\rightarrow_r^{sol}, \rightarrow_w^{sol})$ is a *feasible solution* of $\mathcal{C}$ and T, if $\rightarrow_r^t \subseteq \rightarrow_r^{sol} \subseteq \rightarrow_r$, $\rightarrow_w^t \subseteq \rightarrow_w^{sol} \subseteq \rightarrow_w$ and $\mathcal{C}' = (S, O, \rightarrow_r^{sol}, \rightarrow_w^{sol})$ does not have any threat. The size of a feasible solution Sol, denoted $size(Sol)$, is the value $| \rightarrow_r^{sol} | + | \rightarrow_w^{sol} |$. The *MDFP* is to maximize $size(Sol)$.

3.1 MDFP is NP-complete

Here we show that the decision problem associated to MDFP is NP-complete. Given an instance $I = (\mathcal{C}, T)$ of MDFP and a positive integer K, the decision problem associated to MDFP, called D-MDFP, asks if there is a feasible solution of I of size greater or equal to K.

Theorem 2. *D-MDFP is NP-complete.*

See Appendix 7.2 for the proof.

3.2 ILP Formulation

Here we define a reduction from MDFP to integer linear programming (ILP). In the rest of this section, we denote by $I = (\mathcal{C}, T)$ to be an instance of MDFP, where $\mathcal{C} = (S, O, \rightarrow_r, \rightarrow_w)$ and $T = (\rightarrow_r^t, \rightarrow_w^t)$.

The set of variables $\mathcal{V}$ of the ILP formulation is:

$$\mathcal{V} = \{r_{o,s} \mid o \in O \wedge s \in S \wedge o \rightarrow_r s\} \cup \{w_{s,o} \mid s \in S \wedge o \in O \wedge o \rightarrow_r s\}$$

The domain of the variables in $\mathcal{V}$ is $\{0, 1\}$, and the intended meaning of these variables is the following. Let $\eta_I : \mathcal{V} \rightarrow \{0, 1\}$ be an assignment of the variables in $\mathcal{V}$ corresponding to an optimal solution of the ILP formulation. Then, a solution for I is obtained by removing all permissions corresponding to the variables assigned to 0 by η_I. Formally, $Sol_{\eta_I} = (\rightarrow_r^{sol}, \rightarrow_w^{sol})$ is a solution for I, where

$$\rightarrow_r^{sol} = \{\, (o, s) \mid o \in O \wedge s \in S \wedge o \rightarrow_r s \wedge \eta_I(r_{o,s}) = 1 \,\}$$

$$\rightarrow_w^{sol} = \{\, (s, o) \mid s \in S \wedge o \in O \wedge s \rightarrow_w o \wedge \eta_I(w_{s,o}) = 1 \,\}.$$

The main idea on how we define the ILP encoding, hence its correctness, derives straightforwardly from Theorem 1: we impose that every flow path of length one, say $o \rightarrow_r \hat{s} \rightarrow_w o'$, if these permissions remain in the resulting access control system $\mathcal{C}' = (S, O, \rightarrow_r^{sol}, \rightarrow_w^{sol})$, then it must be the case that for every subject $s \in S$ if s can read from o' in $\mathcal{C}'$, s must also be able to read from o in $\mathcal{C}'$ (Confidentiality), and if s that can write into o in $\mathcal{C}'$, s must be also able to write into o' in $\mathcal{C}'$ (Integrity). Formally, the linear equations of our ILP formulation is the minimal set containing the following.

Confidentiality Constraints: For every sequence of the form $o \rightarrow_r \hat{s} \rightarrow_w \hat{o} \rightarrow_r s$, we add the constraint: $r_{o,\hat{s}} + w_{\hat{s},\hat{o}} + r_{\hat{o},s} - G \leq 2$ where G is $r_{o,s}$ in case $o \rightarrow_r s$, otherwise $G = 0$. For example, for the sequence $o_1 \rightarrow_r s_1 \rightarrow_w o_3 \rightarrow_r s_2$, in the ACS depicted in Fig. 1(a), we have $r_{o_1,s_1} + w_{s_1,o_3} + r_{o_3,s_2} - 0 \leq 2$.

Integrity Constraints: For every sequence of the form $s \rightarrow_w o \rightarrow_r \hat{s} \rightarrow_w \hat{o}$, we add the constraint: $w_{s,o} + r_{o,\hat{s}} + w_{\hat{s},\hat{o}} - G \leq 2$ where G is $w_{s,\hat{o}}$ in case $s \rightarrow_w \hat{o}$, otherwise $G = 0$. As above, for the sequence $s_2 \rightarrow_w o_4 \rightarrow_r s_3 \rightarrow_w o_7$, in the ACS depicted in Fig. 1(a), we add the constraint $w_{s_2,o_4} + r_{o_4,s_3} + w_{s_3,o_7} - 0 \leq 2$.

Trusted Read Constraints: For every $o \rightarrow_r^t s$, we have the constraint: $r_{o,s} = 1$.

Trusted Write Constraints: For every $s \rightarrow_w^t o$, we have the constraint: $w_{s,o} = 1$.

It is easy to see that any variable assignment η that obeys all linear constraints defined above leads to a feasible solution of I.

Objective Function: Now, to maximize the number of remaining permissions (or equivalently, minimize the number of removed permissions) we define the objective function of the ILP formulation as the sum of all variables in $\mathcal{V}$. Compactly, our ILP-FORMULATION$(\mathcal{C}, T)$ is as shown in Fig. 2.

$$
\begin{aligned}
& max && \sum_{v \in \mathcal{V}} v \\
& subject\ to && \\
& r_{o,\hat{s}} + w_{\hat{s},\hat{o}} + r_{\hat{o},s} - r_{o,s} && \leq 2, \forall\ o \rightarrow_r \hat{s} \rightarrow_w \hat{o} \rightarrow_r s, o \rightarrow_r s \\
& r_{o,\hat{s}} + w_{\hat{s},\hat{o}} + r_{\hat{o},s} && \leq 2, \forall\ o \rightarrow_r \hat{s} \rightarrow_w \hat{o} \rightarrow_r s, o \not\rightarrow_r s \\
& w_{s,\hat{o}} + r_{\hat{o},\hat{s}} + w_{\hat{s},o} - w_{s,o} && \leq 2, \forall\ s \rightarrow_w \hat{o} \rightarrow_r \hat{s} \rightarrow_w o, s \rightarrow_w o \\
& w_{s,\hat{o}} + r_{\hat{o},\hat{s}} + w_{\hat{s},o} && \leq 2, \forall\ s \rightarrow_w \hat{o} \rightarrow_r \hat{s} \rightarrow_w o, s \not\rightarrow_w o \\
& r_{o,s} = 1, \forall\ o \rightarrow_r^t s; \quad w_{s,o} = 1, \forall\ s \rightarrow_w^t o; \quad v \in \{0,1\}, \forall v \in \mathcal{V} &&
\end{aligned}
$$

Fig. 2. ILP formulation of MDFP.

We now formally state the correctness of our ILP approach, which is entailed from the fact that we remove the minimal number of permissions from $\mathcal{C}$ resulting in a new ACS that does not have any threat of length one, hence from Theorem 1 does not have any threat at all.

Theorem 3. *For any instance I of MDFP, if η_I is an optimal solution of* ILP-FORMULATION(I) *then Sol_{η_I} is an optimal solution of I.*

We note that while the ILP formulation gives the optimal solution, solving two subproblems (one for confidentiality followed by the one for integrity each with only the relevant constraints) does not give an optimal solution.

For example, for the ACS depicted in Fig. 1(a), if we only eliminate the 15 confidentiality vulnerabilities, the optimal solution is to revoke 5 permissions ($o_1 \rightarrow_r s_1$, $o_2 \rightarrow_r s_1$, $o_1 \rightarrow_r s_2$, $o_2 \rightarrow_r s_2$, and $o_6 \rightarrow_r s_5$). This eliminates all of the confidentiality, while all of the original integrity vulnerabilities still exist. No new vulnerabilities are added. Now, if the integrity vulnerabilities are to be eliminated, the optimal solution is to revoke 4 permissions ($s_3 \rightarrow_w o_6$, $s_3 \rightarrow_w o_7$, $s_4 \rightarrow_w o_6$, $s_4 \rightarrow_w o_7$). Thus, the total number of permissions revoked is 9. However, if both confidentiality and integrity vulnerabilities are eliminated together (using the composite ILP in Fig. 2), the optimal solution is to simply revoke 6 permissions ($o_3 \rightarrow_r s_3$, $o_4 \rightarrow_r s_3$, $o_5 \rightarrow_r s_3$, $o_3 \rightarrow_r s_4$, $o_4 \rightarrow_r s_4$, $o_5 \rightarrow_r s_4$), which is clearly lower than 9.

3.3 Compact ILP Formulation

We now present an improved encoding that extends the ILP formulation described in Sect. 3.2 by merging subjects and objects that have the same permissions. This allows us to get a much reduced encoding, in terms of variables, with better performances in practice (see Sect. 5).

Equivalent Subjects: For an instance $I = (\mathcal{C}, T)$ of MDFP with $\mathcal{C} = (S, O, \rightarrow_r, \rightarrow_w)$ and $T = (\rightarrow_r^t, \rightarrow_w^t)$, two subjects are *equivalent* if they have the same permissions. Formally, for a subject $s \in S$, let $read_I(s)$ (respectively, $read_I^t(s)$) denote the set of all objects that can be read (respectively, trust read) by s in $\mathcal{C}$, i.e., $read_I(s) = \{o \in O \mid o \rightarrow_r s\}$ (respectively, $read_I^t(s) = \{o \in O \mid o \rightarrow_r^t s\}$). Similarly, we define $write_I(s) = \{o \in O \mid s \rightarrow_w o\}$ and $write_I^t(s) = \{o \in O \mid s \rightarrow_w^t o\}$. Then, two subjects s_1 and s_2 are *equivalent*, denoted $s_1 \approx s_2$, if $read_I(s_1) read_I(s_2)$, $read_I^t(s_1) = read_I^t(s_2)$, $write_I(s_1) = write_I(s_2)$, and $write_I^t(s_1) = write_I^t(s_2)$.

For every $s \in S$, $[s]$ is the equivalence class of s w.r.t. $\approx$. Moreover, $S^{\approx}$ denotes the quotient set of S by $\approx$. Similarly, we can define the same notion of equivalent objects, with $[o]$ denoting the the equivalence class of $o \in O$, and $O^{\approx}$ denoting the quotient set of O by $\approx$.

Given a read relation $\rightarrow_r \subseteq O \times S$ and two subjects $s_1, s_2 \in S$, $\rightarrow_r [s_1/s_2]$ is a new read relation obtained from $\rightarrow_r$ by assigning to s_2 the same permissions that s_1 has in $\rightarrow_r$: $\rightarrow_r [s_1/s_2] = \quad (\rightarrow_r \setminus (O \times \{s_2\})) \cup \{(o, s_2) \mid o \in O \wedge o \rightarrow_r s_1\}$.

Similarly, $\rightarrow_w [s_1/s_2] = \quad (\rightarrow_w \setminus (\{s_2\} \times O)) \cup \{(s_2, o) \mid o \in O \wedge s_1 \rightarrow_w o\}$. A similar substitution can be defined for objects.

The following lemma states that for any given optimal solution of I it is always possible to derive a new optimal solution in which two equivalent subjects have the same permissions.

Lemma 1. *Let $I = (\mathcal{C}, T)$ be an instance of the MDFP problem, s_1 and s_2 be two equivalent subjects of I, and $Sol' = (\rightarrow_r^{sol}, \rightarrow_w^{sol})$ be a optimal solution of I. Then, $Sol'' = (\rightarrow_r^{sol} [s_1/s_2], \rightarrow_w^{sol} [s_1/s_2])$ is also an optimal solution of I.*

See Appendix 7.1 for the proof.

The following property is a direct consequence of Lemma 1.

Corollary 1. *Let $I = (\mathcal{C}, T)$ with $\mathcal{C} = (S, O, \rightarrow_r, \rightarrow_w)$ be an instance of the MDFP problem that admits a solution. Then, there exists a solution $Sol = (\rightarrow_r^{sol}, \rightarrow_w^{sol})$ of I such that for every pair of equivalent subjects $s_1, s_2 \in S$, s_1 and s_2 have the same permissions in $\mathcal{C} = (S, O, \rightarrow_r^{sol}, \rightarrow_w^{sol})$.*

Lemma 1 and Corollary 1 also hold for equivalent objects. Proofs are similar to those provided above and hence we omit them here.

Compact ILP formulation. Corollary 1 suggests a more compact encoding of the MDFP into ILP. From $\mathcal{C}$, we define a new ACS $\mathcal{C}^{\approx}$ by collapsing all subjects and objects into their equivalence classes defined by $\approx$, and by merging permissions consequently (edges of $G_{\mathcal{C}}$). Formally, $\mathcal{C}^{\approx}$ has $S^{\approx}$ as set of subjects and $O^{\approx}$ as set

$$max \sum_{[o]\to_r^{\approx}[s]} \left(|\,[o]\,| \cdot |\,[s]\,| \cdot r_{[o],[s]} \right) \quad + \quad \sum_{[s]\to_w^{\approx}[o]} \left(|\,[s]\,| \cdot |\,[o]\,| \cdot w_{[s],[o]} \right)$$

$$subject\ to$$

$$r_{[o],[\hat{s}]} + w_{[\hat{s}],[\hat{o}]} + r_{[\hat{o}],[s]} - r_{[o],[s]} \leq 2, \forall [o] \to_r^{\approx} [\hat{s}] \to_w^{\approx} [\hat{o}] \to_r^{\approx} [s] \wedge [o] \to_r [s]$$

$$r_{[o],[\hat{s}]} + w_{[\hat{s}],[\hat{o}]} + r_{[\hat{o}],[s]} \leq 2, \forall [o] \to_r^{\approx} [\hat{s}] \to_w^{\approx} [\hat{o}] \to_r^{\approx} [s] \wedge [o] \not\to_r [s]$$

$$w_{[s],[\hat{o}]} + r_{[\hat{o}],[\hat{s}]} + w_{[\hat{s}],[o]} - w_{[s],[o]} \leq 2, \forall [s] \to_w^{\approx} [\hat{o}] \to_r^{\approx} [\hat{s}] \to_w^{\approx} [o] \wedge [s] \to_w [o]$$

$$w_{[s],[\hat{o}]} + r_{[\hat{o}],[\hat{s}]} + w_{[\hat{s}],[o]} \leq 2, \forall [s] \to_w^{\approx} [\hat{o}] \to_r^{\approx} [\hat{s}] \to_w^{\approx} [o] \wedge [s] \not\to_w [o]$$

$$r_{[o],[s]} = 1, \forall [o] \to_r^{t} [s]; \quad w_{[s],[o]} = 1, \forall [s] \to_w^{t} [o]; \quad v \in \{0,1\}, \forall v \in \mathcal{V}^{\approx}$$

Fig. 3. ILP formulation of MDFP based on equivalence classes.

of objects, where the read and write permission sets are defined as follows: $\to_r^{\approx} = \{\ ([o],[s])\ \mid\ o \in O\ \wedge\ s \in S\ \wedge\ o \to_r s\ \}, \to_w^{\approx} = \{\ ([o],[s])\ \mid\ s \in S\ \wedge\ o \in O\ \wedge\ s \to_w o\ \}$. Similarly, we define the trusted permissions of $\mathcal{C}^{\approx}$ as $T^{\approx} = (\to_r^{t\,\approx}, \to_w^{t\,\approx})$ where $\to_r^{t\,\approx} = \{\ ([o],[s])\ \mid\ o \in O\ \wedge\ s \in S\ \wedge\ o \to_r^t s\ \}, \to_w^{t\,\approx} = \{\ ([o],[s])\ \mid\ s \in S\ \wedge\ o \in O\ \wedge\ s \to_w^t o\ \}$.

We now define a new ILP encoding, Compact-ILP-formulation(I), for MFDP on the instance $(\mathcal{C}^{\approx}, T^{\approx})$, which is similar to that of Fig. 2 with the difference that now edges may have a weight greater than one; reflecting the number of edges of $\mathcal{C}$ it represents in $\mathcal{C}^{\approx}$. More specifically, each edge from a node x_1 to x_2 in $G_{\mathcal{C}^{\approx}}$ represents all edges from all nodes in $[x_1]$ to all nodes in $[x_2]$, i.e., its weight is $|[x_1]| \cdot |[x_2]|$. Figure 1(b) shows the compact representation of Fig. 1(a), where the edges have the appropriate weights.

Figure 3 shows Compact-ILP-formulation(I) over the set of variables $\mathcal{V}^{\approx}$. The set of linear constraints is the same as those in Fig. 2 with the difference that now they are defined over $\mathcal{C}^{\approx}$ rather than $\mathcal{C}$. Instead, the objective function is similar to that of Fig. 2, but now captures the new weighting attributed to edges in $G_{\mathcal{C}^{\approx}}$.

Let $\eta_I^{\approx} : \mathcal{V} \to \{0,1\}$ be a solution to the ILP instance of Fig. 3. Define $\widehat{Sol}_{\eta_I^{\approx}} = (\widehat{\to_r}^{sol}, \widehat{\to_w}^{sol})$ where $\widehat{\to_r}^{sol} = \{\ (o,s) \in O \times S\ \mid\ o \to_r s\ \wedge\ \eta_I^{\approx}(r_{[o],[s]}) \geq 1\ \}$ and $\widehat{\to_w}^{sol} = \{\ (s,o) \in S \times O\ \mid\ s \to_w o\ \wedge\ \eta_I^{\approx}(w_{[s],[o]}) \geq 1\ \}$.

We now prove that $Sol_{\eta_I^{\approx}}$ is an *optimal* solution of I.

Theorem 4. *For any instance I of MDFP, if $\eta_I^{\approx}$ is an optimal solution of* Compact-ILP-formulation(I) *then $\widehat{Sol}_{\eta_I^{\approx}}$ is an optimal solution of I. Furthermore, if I admits a solution then $\eta_I^{\approx}$ also exists.*

See Appendix 7.3 for the proof.

4 Preventing Data Leakages with Monitors

A *data-leakage monitor* or simply *monitor* of an access control system $\mathcal{C}$ is a computing system that by observing the behaviors on $\mathcal{C}$ (i.e., the sequence of read

and write operations) detects and prevents data leakages (both confidentiality and integrity violations) by blocking subjects' operations. In this section, we present a monitor based on a tainting approach. We first define monitors as language acceptors of runs of $\mathcal{C}$ that are data leakage free. We then present a monitor based on tainting and then conclude with an optimized version of this monitor that uses only 2-step tainting, leading to better empirical performances.

Monitors. Let $\mathcal{C} = (S, O, \rightarrow_r, \rightarrow_w)$ be an ACS, $\Sigma = S \times \{read, write\} \times O$ be the set of all possible *actions* on $\mathcal{C}$, and $R = \{accept, reject\}$. A *monitor* $\mathcal{M}$ of $\mathcal{C}$ is a triple (Q, q_{st}, δ) where Q is a set of *states*, $q_{st} \in Q$ is the *start state*, and $\delta : (Q \times R \times \Sigma) \rightarrow (Q \times R)$ is a (deterministic) *transition function*.

A *configuration* of $\mathcal{M}$ is a pair (q, h) where $q \in Q$ and $h \in R$. For a word $w = \sigma_1 \dots \sigma_m \in \Sigma^*$ with actions $\sigma_i \in \Sigma$ for $i \in [1, m]$, a *run* of $\mathcal{M}$ on w is a sequence of $m+1$ configurations $(q_0, h_0), \dots (q_m, h_m)$ where q_0 is the start state q_{st}, $h_0 = accept$, and for every $i \in [1, m]$ the following holds: $h_{i-1} = accept$ and $(q_i, h_i) = \delta(q_{i-1}, h_{i-1}, \sigma_i)$, or $h_{i-1} = h_i = reject$ and $q_i = q_{i-1}$.

A word w (run of $\mathcal{C}$) is *accepted* by $\mathcal{M}$ if $h_m = accept$. The *language* of $\mathcal{M}$, denoted $L(\mathcal{M})$, is the set of all words $w \in \Sigma^*$ that are accepted by $\mathcal{M}$.

A monitor $\mathcal{M}$ is *maximal data leakage preserving* (MDLP, for short) if $L(\mathcal{M})$ is the set of all words in Σ^* that are confidentiality and integrity free. For any given ACS $\mathcal{C}$, it is easy to show that an MDLP monitor can be built. This can be proved by showing that $L(\mathcal{M})$ is a regular language: we can easily express the properties of the words in $L(\mathcal{M})$ with a formula φ of monadic second order logic (MSO) on words and then use an automatic procedure to convert φ into a finite state automaton [14]. Although, this is a convenient way of building monitors for regular properties, it can lead to automata of exponential size in the number of objects and subjects. Hence, it is not practical for real access control systems.

Building Maximal Data-Leakage Preserving Monitors. A monitor based on tainting can be seen as a dynamic information flow tracking system that is used to detect data flows (see for example [17,21,22]).

An MDLP monitor $\mathcal{M}_{taint}$ based on tainting associates each subject and object with a subset of subjects and objects (*tainting sets*). $\mathcal{M}_{taint}$ starts in a state where each subject and object is tainted with itself. Then, $\mathcal{M}_{taint}$ progressively scans the sequence of actions on $\mathcal{C}$. For each action, say from an element x_1 to an element x_2, $\mathcal{M}_{taint}$ updates its state by propagating the tainting from x_1 to x_2. These tainting sets can be seen as a way to represent the endpoints of all flows: if x_2 is tainted by x_1, then there is a flow from x_1 to x_2. Thus, by using these flows and the definitions of confidentiality and integrity violations, $\mathcal{M}_{taint}$ detects data leakages.

More formally, an $\mathcal{M}_{taint}$ state is a map $taint : (S \cup O) \rightarrow 2^{(S \cup O)}$. A state $taint$ is a start state if $taint(x) = \{x\}$, for every $x \in (S \cup O)$. The transition relation δ of $\mathcal{M}_{taint}$ is defined as follows. For any two states $taint, taint', h, h' \in R$ and $\sigma = (s, op, o) \in \Sigma$, $\delta(taint, h, \sigma) = (taint', h')$ if either $h = h' = reject$ and $taint' = taint$, or $h = accept$ and the following holds:

(Data Leakage) $h' = \mathit{reject}$ iff either (Confidentiality Violation) $op = \mathit{read}$ and $\exists \hat{o} \in \mathit{taint}(o)$ such that $\hat{o} \not\rightarrow_r s$, or (Integrity Violation) $op = \mathit{write}$ and $\exists \hat{s} \in \mathit{taint}(s)$ such that $\hat{s} \not\rightarrow_w o$.

(Taint Propagation) either (Read Propagation) $op = \mathit{read}$, $\mathit{taint}'(s) = (\mathit{taint}(s) \cup \mathit{taint}(o))$, and for every $x \in (S \cup O) \setminus \{s\}$, $\mathit{taint}'(s) = \mathit{taint}(s)$; or (Write Propagation) $op = \mathit{write}$, $\mathit{taint}'(o) = (\mathit{taint}(o) \cup \mathit{taint}(s))$, and for every $x \in (S \cup O) \setminus \{o\}$, $\mathit{taint}'(x) = \mathit{taint}(s)$.

Theorem 5. *$\mathcal{M}_{taint}$ is an MDLP monitor.*

MDLP Monitor Based on 2-Step Tainting: The tainting sets of $\mathcal{M}_{taint}$ progressively grow as more flows are discovered. In the limit each tainting set potentially includes all subjects and objects of $\mathcal{C}$. Since for each action the time for checking confidentiality and integrity violations is proportional to the size of the tainting sets of the object and subject involved in that action, it is desirable to reduce the sizes of these sets to get better performances. We achieve this, by defining a new tainting monitor $\mathcal{M}^2_{taint}$ that keeps track only of the flows that across at most two adjacent edges in $G_{\mathcal{C}}$. The correctness of our construction is justified by the correctness of $\mathcal{M}_{taint}$ and Theorem 1.

The 2-step tainting monitor $\mathcal{M}^2_{taint}$ is defined as follows. A state of $\mathcal{M}^2_{taint}$ is (as for $\mathcal{M}_{taint}$) a map $\mathit{taint} : (S \cup O) \to 2^{(S \cup O)}$. Now, a state taint is a start state if $\mathit{taint}(x) = \emptyset$, for every $x \in (S \cup O)$.

The transition relation δ^2 of $\mathcal{M}^2_{taint}$ is defined to guarantee that after reading a violation free run π of $\mathcal{C}$:

- for every $s \in S$, $x \in \mathit{taint}(s)$ iff either (1) $x \in O$, (o, s) is an edge of $G_{\mathcal{C}}$, and there is a direct flow from x to s in π, or (2) $x \in S$, for some subject $\hat{o} \in O$, $(x, \hat{o}, s)$ is a path in $G_{\mathcal{C}}$, and there is a 2-step flow from x to s in π;
- for every $o \in O$, $x \in \mathit{taint}(o)$ iff either (1) $x \in S$, (s, o) is an edge of $G_{\mathcal{C}}$, and there is a direct flow from x to o in π, or (2) $x \in O$, for some subject $\hat{s} \in S$, $(x, \hat{s}, o)$ is a path in $G_{\mathcal{C}}$, and there is a 2-step flow from x to o in π.

Formally, for any two states $\mathit{taint}, \mathit{taint}'$, $h, h' \in R$ and $\sigma = (s, op, o) \in \Sigma$, $\delta^2(\mathit{taint}, h, \sigma) = (\mathit{taint}', h')$ if either $h = h' = \mathit{reject}$ and $\mathit{taint}' = \mathit{taint}$, or $h = \mathit{accept}$ and the following holds:

(Data Leakage) same as for $\mathcal{M}_{taint}$;

(Taint Propagation) either (Read Propagation) $op = \mathit{read}$, $\mathit{taint}'(s) = \mathit{taint}(s) \cup \{o\} \cup (\mathit{taint}(o) \cap S)$, and for every $x \in (S \cup O) \setminus \{s\}$, $\mathit{taint}'(s) = \mathit{taint}(s)$; or (Write Propagation) $op = \mathit{write}$, $\mathit{taint}'(o) = \mathit{taint}(o) \cup \{s\} \cup (\mathit{taint}(s) \cap O)$, and for every $x \in (S \cup O) \setminus \{o\}$, $\mathit{taint}'(x) = \mathit{taint}(s)$.

From the definition of $\mathcal{M}^2_{taint}$ it is simple to show (by induction) that the following property holds.

Theorem 6. *$\mathcal{M}^2_{taint}$ is an MDLP monitor. Furthermore, for every $\mathcal{C}$ run $\pi \in \Sigma^*$, if $(\mathit{taint}_0, h_0), \ldots (\mathit{taint}_m, h_m)$ and $(\mathit{taint}'_0, h'_0), \ldots (\mathit{taint}'_m, h'_m)$ are, respectively, the run of $\mathcal{M}_{taint}$ and $\mathcal{M}^2_{taint}$ on π, then $\mathit{taint}'_i(x) \subseteq \mathit{taint}_i(x)$, for every $i \in [1, m]$ and $x \in (S \cup O)$.*

Therefore, in practice we expect that for large access control systems $\mathcal{M}^2_{taint}$ is faster than $\mathcal{M}_{taint}$ as each tainting sets of $\mathcal{M}^2_{taint}$ will be local and hence much smaller in size than those of $\mathcal{M}_{taint}$. To show the behavior of the monitor the based approach, consider again the access control system shown in Table 1, along with the potential sequence of operations shown in Table 2. Table 2 shows the taints and monitor's action for each operation in the sequence. Note that the monitor blocks a total of six permissions (2 each on operations (2), (3), and (5)).

Table 2. Sample sequence of actions and monitor's behavior

	User's operation	Actions taken
1	s_1, r, o_1	$taint(s_1) = \{o_1\}$
2	s_1, w, o_3	$taint(o_3) = \{s_1, o_1\}$ Monitor will block $o_3 \rightarrow_r s_3$ $o_3 \rightarrow_r s_4$ to remove the confidentiality vulnerabilities
3	s_1, w, o_4	$taint(o_4) = \{s_1, o_1\}$ Monitor will block $o_4 \rightarrow_r s_3$ $o_4 \rightarrow_r s_4$ to remove the confidentiality vulnerabilities
4	s_2, w, o_4	$taint(o_4) = \{s_1, o_1, s_2\}$
5	s_4, r, o_4	$taint(s_4) = \{s_1, s_2, o_4\}$ Monitor will block $s_4 \rightarrow_w o_6$ and $s_4 \rightarrow_w o_7$ to remove the integrity vulnerability
6	s_3, r, o_3	Access denied
7	s_4, w, o_7	Access denied

5 Experimental Evaluation

We now present the experimental evaluation which demonstrates the performance and restrictiveness of the two proposed approaches. We utilize four real life access control data sets with users and permissions – namely, (1) fire1, (2) fire2, (3) domino, (4) hc [12]. Note that these data sets encode a simple access control matrix denoting the ability of a subject to access an object (in any access mode). Thus, these data sets do not have the information regarding which particular permission on the object is granted to the subject. Therefore, we assume for all of the datasets that each assignment represents both a read and a write permission on a distinct object.

For the data leakage free by design approach, we use the reduced access control matrices obtained by collapsing equivalent subjects and objects, as discussed in Sect. 3. The number of subjects and objects in the original and reduced matrices are given in Table 3. Note that collapsing subjects and objects significantly reduces the sizes of the datasets (on average the dataset is reduced by 93.99%). Here, by size, we mean the product of the number of subjects and objects. Since the number of constraints is linearly proportional to the number of permissions which depends on the number of subjects and objects, a reduction in their size leads to a smaller ILP problem.

Table 3. Dataset details

Dataset	Name	Original size		Reduced size		Percentage
		Subjects	Objects	Subjects	Objects	Reduction
1	fire1	365	709	90	87	96.97 %
2	fire2	325	590	11	11	99.94 %
3	domino	79	231	23	38	95.21 %
4	hc	46	46	18	19	83.84 %

We implement the solution approaches described above. For the data leakage free by design approach (Sect. 3), we create the appropriate ILP model as per Fig. 3. The ILP model is then executed using IBM CPLEX (v 12.5.1) running through callable libraries within the code. For the monitor based approach, the $\mathcal{M}^2_{taint}$ monitor is implemented. The algorithms are implemented in C and run on a Windows machine with 16 GB of RAM and Core i7 2.93 GHz processor.

Table 4 presents the experimental results for the Data Leakage Free by Design approach. The column "Orig. CPLEX Time", shows the time required to run the ILP formulation given in Fig. 2, while the column "Red. CPLEX Time" gives the time required to run the compact ILP formulation given in Fig. 3. As can be seen, the effect of collapsing the subjects and objects is enormous. fire1 and fire2 could not be run (CPLEX gave an out of memory error) for the original access control matrix, while the time required for hc and domino was several orders of magnitude more. Since we use the reduced datasets, as discussed above, the column "Threats" reflects the number of threats in the reduced datasets to be eliminated. The next three columns depict the amount of permission revocation to achieve a data leakage free access matrix. Note that, here we list the number of permissions revoked in the original access control matrix. On average, 25.28% of the permissions need to be revoked to get an access control system without any data leakages.

When we have a monitor, as discussed in Sect. 4, revocations can occur on the fly. Therefore, to test the relative performance of the monitor based approach, we have randomly generated a set of read/write operations that occur in the order they are generated. The monitor based approach is run and the number of

Table 4. Results for data leakage free access matrix

Dataset	Orig. CPLEX Time (s)	Red. CPLEX Time (s)	Threats	# Perm. Init. Assn	# Perm. Revoked	% Revoked
1	-	2582	34240	63902	14586	22.83 %
2	-	0.225	514	72856	12014	16.49 %
3	8608.15	6.01	3292	1460	421	28.84 %
4	1262.82	0.27	1770	2972	980	32.97 %

Table 5. Results for monitor based approach

Dataset	# Perm. Init. Assn.	Number permissions blocked						% Finally blocked
		10%	50%	100%	1000%	5000%	10000%	
1	63902	0	140	532	14221	24031	26378	41.28 %
2	72856	0	13	26	3912	8129	9025	12.39 %
3	1460	0	36	41	130	283	364	24.93 %
4	2972	0	0	0	557	1123	1259	42.36 %

permissions revoked is counted. Since the number of flows can increase as more operations occur, and therefore lead to more revocations, we actually count the revocations for a varying number of operations. Specifically, for each dataset, we generate on average 100 operations for every subject (i.e., we generate $100 * |S|$ number of random operations). Thus, for hc, since there are 46 subjects, we generate 4600 random operations, where as for fire1 which has 365 subjects, we generate 36500 random operations. Now, we count the number of permissions revoked if only $10\% * |S|$ operations are carried out (and similarly for $50\% * |S|$, $100\% * |S|$, $1000\% * |S|$, $5000\% * |S|$, and finally $10000\% * |S|$). Table 5 gives the results. Again, we list the number of permissions revoked in the original access control matrix. As we can see, the number of permissions revoked is steadily increasing, and in the case of fire1 and hc the final number of permissions revoked is already larger than the permissions revoked in the data leakage free method. Also, note that in the current set of experiments, we have set a window size of 1000 – this means that if the gap between a subject reading an object and then writing to another object is more than 1000 operations, then we do not consider a data flow to have occurred (typically a malicious software would read and then write in a short duration of time) – clearly, the choice of 1000 is arbitrary, and in fact, could be entirely removed, to ensure no data leakages. In this case, the number of permission revocations would be even larger than what is reported, thus demonstrating the benefit of the data leakage free approach when a large number of operations are likely to be carried out.

6 Related Work

The importance of preventing inappropriate leakage of data, often called the confinement problem in computer systems, first identified by Lampson in early 70's [20], is defined as the problem of assuring the ability to limit the amount of damage that can be done by malicious or malfunctioning software. The need for a confinement mechanism first became apparent when researchers noted an important inherent limitation of DAC – the Trojan Horse Attack, and with the introduction of the Bell and LaPadula model and the MAC policy. Although MAC compliant systems prevent inappropriate leakage of data, these systems are limited to multi-level security.

While MAC is not susceptible to Trojan Horse attacks, many solutions proposed to prevent any such data leakage exploit employing labels or type based access control. Boebert et al. [3], Badger et al. [1] and Boebert and Kain [4] are some of the studies that address confidentiality violating data flows. Mao et al. [21] propose a label based MAC over a DAC system. The basic idea of their approach is to associate read and write labels to objects and subjects. These object labels are updated dynamically to include the subject's label when the subject reads or writes to that object. Moreover, the object label is a monotonically increasing set of items, with the cardinality in the order of the number of users read (wrote) the object. Their approach detects integrity violating data flows. Zimmerman et al. [32] propose a rule based approach that prevents any integrity violating data flow. Jaume et al. [17] propose a dynamic label updating procedure that detects if there is any confidentiality violating data flow.

Information Flow Control (IFC) models [10,18] are closely related to our problem. IFC model is a fine-grained information flow model which is also based on tainting and utilizes labels for each piece of data that is required to be protected using the lattice model for information flow security by [9]. The models can be at software or OS level depending on the granularity of the control and centralized or decentralized depending on the authority to modify labels [24]. However, these models do not consider the permission assignments, which makes them different than our model.

Dynamic taint analysis is also related to our problem. Haldar et al. [16] propose a taint based approach for programs in Java, and Lam et al. [19] propose a dynamic taint based analysis on C. Enck et al. [11] provide a taint based approach to track third party Android applications. Cheng et al. [6], Clause et al. [7] and Zhu et al. [31] propose software level dynamic tainting.

Sze et al. [26] study the problem of self-revocation, where a revocation in the permission assignments of any subject on an object while editing it might cause confidentiality and integrity issues. They also study the problem of integrity violation by investigating the source code and data origin of suspected malware and prevent any process that is influenced from modifying important system resources [27]. Finally, the work by Gong and Qian [15] focuses on detecting the cases where confidentiality and integrity flows occur due to interoperation of distinct access control systems. They study the complexity to detect such violations.

7 Conclusions and Future Work

In this paper, we have proposed a methodology for identifying and eliminating unauthorized data flows in DAC, that occur due to Trojan Horse attacks. Our key contribution is to show that a transitive closure is not required to eliminate such flows. We then propose two alternative solutions that fit different situational needs. We have validated the performance and restrictiveness of the proposed approaches with real data sets. In the future, we plan to propose an auditing based approach which eliminates unauthorized flows only if the flows

get realized. This might be useful to identify the data leakage channels that are actually utilized. We also plan to extend our approach to identify and prevent the unauthorized flows in RBAC, which is also prone to Trojan Horse attacks. Analysis on RBAC is more challenging since there is an additional layer of complexity (roles) that must be taken into account. The preventive action decisions must overcome the dilemma of whether to revoke the role from the user or revoke the permission from the role.

Appendix

7.1 Proof of Lemma 1

Proof. Assume that S and O are the set of subjects and objects of $\mathcal{C}$, respectively. Let $\mathcal{C}' = (S, O, \rightarrow_r^{sol}, \rightarrow_w^{sol})$ and $\mathcal{C}'' = (S, O, \rightarrow_r^{sol}\ [s_1/s_2], \rightarrow_w^{sol}\ [s_1/s_2])$.

We first prove (by contradiction) that Sol'' is a **feasible** solution of I. Assume that $\mathcal{C}''$ has a threat. This threat is witnessed by a flow path, say ρ, that must contain s_2. If ρ does not involve s_2 then ρ would also be a threat in $\mathcal{C}'$, which cannot be true as Sol' is a feasible solution of I. Now, observe that s_2 can always be replaced by s_1 along any flow path of $\mathcal{C}''$, as s_2 and s_1 have the same neighbor in $G_{\mathcal{C}''}$. Thus, the flow path obtained by replacing s_2 with s_1 along ρ, also witnesses a threat in $\mathcal{C}'$. Again a contradiction. Therefore, Sol'' is a feasible solution of I.

We now prove that Sol'' is also **optimal** (that is, $size(Sol') = size(Sol'')$) by showing that s_1 and s_2 have the same number of incident edges in $G_{\mathcal{C}'}$. Let n_1 (respectively, n_2) be the number of incident nodes of s_1 (respectively, s_2) in $G_{\mathcal{C}'}$. By contradiction, and w.l.o.g., assume that $n_1 > n_2$. Since $\mathcal{C}''$ is obtained from $\mathcal{C}'$ by removing first the permissions of s_2 and then adding to s_2 the same permissions of s_1, it must be the case that $size(Sol'') > size(Sol')$. This would entail that Sol' is not an optimal solution, which is a contradiction.

7.2 Proof of Theorem 2

NP-membership. Let $Sol = (\rightarrow'_r, \rightarrow'_w)$ such that $\rightarrow'_r, \rightarrow'_w \subseteq S \times O$. To check whether Sol is a feasible solution of I, we need to check that (1) $\rightarrow_r^t \subseteq \rightarrow'_r \subseteq \rightarrow_r$, (2) $\rightarrow_w^t \subseteq \rightarrow'_w \subseteq \rightarrow_w$, (3) $\mid \rightarrow'_r \mid + \mid \rightarrow'_w \mid \geq K$, and more importantly, (4) that $(S, O, \rightarrow'_r, \rightarrow'_w)$ is an ACS that does not contain any threat. The first three properties are easy to realize in polynomial time. Concerning the last property, we exploit Theorem 1. To check that there is no confidentiality threat, we build all sequences of the form $o_0 \rightarrow'_r s_0 \rightarrow'_w o_1 \rightarrow'_r s_1$ and then verify the existence of the read permission $o_0 \rightarrow'_r s_1$. Similarly, for integrity threat we build all sequences such that $s_0 \rightarrow'_w o_0 \rightarrow'_r s_0 \rightarrow'_w o_1$ and then check the existence of the write permission $s_0 \rightarrow'_w o_1$. Note that, all these sequences can be built in $O(O^2 \cdot S^2)$ and these checks can all be accomplished in polynomial time. This shows that D-MDFP belongs to NP.

NP-hardness. For the NP-hardness proof, we provide a polynomial time reduction from *the edge deletion transitive digraph* problem (ED-TD) to D-MDFP.

The ED-TD asks to remove the minimal number of edges from a given directed graph such that the resulting graph corresponds to its transitive closure. ED-TD problem is known to be NP-complete (see [28] Theorem 15, and [29]).

The reduction is as follows. Let $G = (V, E)$ be a directed graph with set of nodes $V = \{1, 2, \ldots n\}$ and set of edges $E \subseteq (V \times V)$. We assume that nodes of G do not have self-loops. We now define the instance $I_G = (\mathcal{C}_G, T_G)$ of D-MDFP to which G is reduced to. Let $\mathcal{C}_G = (S, O, \rightarrow_r, \rightarrow_w)$ and $T_G = (\rightarrow_r^t, \rightarrow_w^t)$. $\mathcal{C}_G$ has a subject s_i and an object o_i, for each node $i \in V$. Moreover, there is a read permission from o_i to s_i, and a write permission from s_i to o_i, for every node $i \in V$. These permissions are also trusted, i.e., belonging to $\rightarrow_r^t$ and $\rightarrow_w^t$, respectively; and no further permissions are trusted. Furthermore, for every edge $(i, j) \in E$, there is a read permission from o_i to s_j, and a write permission from s_i to o_j. Formally, $S = \{s_i \mid i \in V\}$ and $O = \{o_i \mid i \in V\}$; $\rightarrow_r^t = \{(o_i, s_i) \mid i \in V\}$; $\rightarrow_w^t = \{(s_i, o_i) \mid i \in V\}$; $\rightarrow_r = \rightarrow_r^t \cup \{(o_i, s_j) \mid (i, j) \in E\}$; $\rightarrow_w = \rightarrow_w^t \cup \{(s_i, o_j) \mid (i, j) \in E\}$.

Lemma 2. *Let G be a directed graph with nodes $V = \{1, 2, \ldots, n\}$, and $Sol = (\rightarrow'_r, \rightarrow'_w)$ be a feasible solution of I_G. For any $i, j \in V$ with $i \neq j$, $o_i \rightarrow'_r s_j$ if and only if $s_i \rightarrow'_w o_j$.*

Proof. The proof is by contradiction. Consider first the case when $o_i \rightarrow'_r s_j$ and $s_i \not\rightarrow'_w o_j$. Observe that, $s_i \rightarrow'_w o_i$ and $s_j \rightarrow_w o_j$ exist as both of them are trusted permissions of I_G. Thus, $s_i \rightarrow'_w o_i \rightarrow'_r s_j \rightarrow_w o_j$ is an integrity threat, leading to a contradiction. The case when $o_i \not\rightarrow'_r s_j$ and $s_i \rightarrow'_w o_j$ is symmetric, and we omit it here.

We now show that the transformation defined above from G to I_G is indeed a polynomial reduction from ED-TD to D-MDFP. The NP-hardness directly follows from the following lemma.

Lemma 3. *Let G be a directed graph with n nodes. G contains a subgraph G' with K edges whose transitive closure is G' itself if and only I_G admits a feasible solution Sol of size $2 \cdot (n + K)$.*

Proof. Let $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, $G' = (V, E')$, $I_G = (\mathcal{C}_G, T_G)$ where $\mathcal{C}_G = (S, O, \rightarrow_r, \rightarrow_w)$ and $T_G = (\rightarrow_r^t, \rightarrow_w^t)$, and $Sol = (\rightarrow'_r, \rightarrow'_w)$.

"only if" direction. Assume that G' is the transitive closure of itself and $|E'| = K$. We define *Sol* as follows: $\rightarrow'_r = \rightarrow_r^t \cup \{o_i \rightarrow_r s_j \mid (i, j) \in E'\}$ and $\rightarrow'_w = \rightarrow_w^t \cup \{s_i \rightarrow_w o_j \mid (i, j) \in E'\}$. From the definition of I_G, it is straightforward to see that $size(Sol) = 2 \cdot (n + K)$. To conclude the proof we only need to show that *Sol* is a feasible solution of I_G. Since $\rightarrow_r^t \subseteq \rightarrow'_r$ and $\rightarrow_w^t \subseteq \rightarrow'_w$ we are guaranteed that *Sol* contains all trusted permissions of T_G. We now show that $\mathcal{C}' = (S, O, \rightarrow'_r, \rightarrow'_w)$ does not contain any threat. Assume that there is a threat in $\mathcal{C}'$. By Theorem 1, there must be a threat of length one. If it is a confidentiality threat, then $o_i \rightarrow'_r s_k \rightarrow'_w o_z \rightarrow'_r s_j$ and $o_i \not\rightarrow'_r s_j$, for some $i, k, z, j \in V$ with $i \neq j$. From the definition of I_G, it must be the case that there is a path from node i to node j in G'

and $(i', j) \notin E$ which leads to a contradiction. The case of integrity vulnerabilities is symmetric.

"if" direction. Assume that *Sol* is a feasible solution of I_G of size $2 \cdot (n + K)$. We define $E' = \{(i, j) \mid i \neq j \wedge o_i \rightarrow'_r s_j\}$. Note that, in the definition of E' using permission $s_i \rightarrow'_w o_j$ rather than $o_i \rightarrow'_r s_j$ would lead to the same set of edges E' (see Lemma 2). By the definition of I_G and Lemma 2, it is direct to see the G' is a subgraph of G and $|E'| = K$. We now show that the transitive closure of G' is again G'. By contradiction, assume that there is a path from node i to node j in G' and there is no direct edge from i to j. But this implies that in the access control system $(S, O, \rightarrow'_r, \rightarrow'_w)$ there is a sequence of alternating read and write operations from object o_i to subject s_j and $o_i \not\rightarrow'_r s_j$, which witnesses a confidentiality threat. This is a contradiction as *Sol* is a feasible solution of I_G.

7.3 Proof of Theorem 4

Proof. Let $I = (\mathcal{C}, T)$, $\widehat{Sol}_{\eta_I^{\approx}} = (\widehat{\rightarrow_r}^{sol}, \widehat{\rightarrow_w}^{sol})$, $\mathcal{C}' = (S, O, \widehat{\rightarrow_r}^{sol}, \widehat{\rightarrow_w}^{sol})$, and $\mathcal{C}^{\approx} = (S^{\approx}, O^{\approx}, \rightarrow_r^{\approx}, \rightarrow_w^{\approx})$. We first show that $\widehat{Sol}_{\eta_I^{\approx}}$ is a **feasible** solution of I. Assume by contradiction that $\mathcal{C}'$ has a one-step confidentiality threat, say $o \rightarrow_r^{sol} \hat{s} \rightarrow_w^{sol} o' \rightarrow_r^{sol} s \wedge o \not\rightarrow_r^{sol} s$. It is easy to see that $[o] \rightarrow_r^{\approx} [\hat{s}] \rightarrow_w^{\approx} [o'] \rightarrow_r^{\approx} [s] \wedge [o] \not\rightarrow_r^{\approx} [s]$ holds, but this is not possible since Compact-ILP-formulation(I) contains a constraint that prevents that these relations hold conjunctly. A similar proof exists for integrity vulnerabilities. Therefore, $\widehat{Sol}_{\eta_I^{\approx}}$ is a feasible solution of I.

Now, we show that $\widehat{Sol}_{\eta_I^{\approx}}$ is also *optimal*. Assume by contradiction that $\widehat{Sol}_{\eta_I^{\approx}}$ is not optimal, and $Sol = (\rightarrow_r^{sol}, \rightarrow_w^{sol})$ is an optimal solution of I where all equivalent subjects/objects have the same permissions. The existence of *Sol* is guaranteed by Corollary 1. Now, we reach a contradiction showing that η_I is not optimal for Compact-ILP-formulation(I). For every $s \in S, o \in O$, $\eta(r_{[o],[s]}) = 1$ (respectively, $\eta(w_{[s],[o]}) = 1$) if and only if $o \rightarrow_r^{sol} s$ (respectively, $s \rightarrow_w^{sol} o$) holds. Notice that η is well defined because all subjects/objects in the same equivalent class have the same permissions in *Sol*. It is straightforward to prove that η allows to satisfy all linear constraints of Compact-ILP-formulation(I), and more importantly leads to a greater value of the objective function. Note that, for the variable assignment η the objective function has a value $n_\eta = size(Sol)$ whereas has value $n_{\eta_I} = size(\widehat{Sol}_{\eta_I^{\approx}})$ for the assignment $\eta_I^{\approx}$. Now, $n_\eta > n_{\eta_I}$, and it cannot be true because $\eta_I^{\approx}$ is an optimal assignment. The definition of η and the fact that it satisfies all linear constraints shows that if I admits a solution then it shows that Compact-ILP-formulation(I) admits a solution. Therefore, $\eta_I^{\approx}$ also exists.

References

1. Badger, L., Sterne, D.F., Sherman, D.L., Walker, K.M., Haghighat, S.A.: Practical domain and type enforcement for UNIX. In: IEEE S&P, pp. 66–77 (1995)
2. Bell, D.E., LaPadula, L.J.: Secure computer systems: mathematical foundations. Technical report, DTIC Document (1973)
3. Boebert, W., Young, W., Kaln, R., Hansohn, S.: Secure ADA target: issues, system design, and verification. In: IEEE S&P (1985)
4. Boebert, W.E., Kain, R.Y.: A further note on the confinement problem. In: Proceedings Security Technology, pp. 198–202. IEEE (1996)
5. Brucker, A.D., Petritsch, H.: Extending access control models with break-glass. In: SACMAT, pp. 197–206. ACM (2009)
6. Cheng, W., Zhao, Q., Yu, B., Hiroshige, S.: Tainttrace: efficient flow tracing with dynamic binary rewriting. In: ISCC, pp. 749–754. IEEE (2006)
7. Clause, J., Li, W., Orso, A.: Dytan: a generic dynamic taint analysis framework. In: ISSTA, pp. 196–206. ACM (2007)
8. Crampton, J.: Cryptographic enforcement of role-based access control. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 191–205. Springer, Heidelberg (2011)
9. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
10. Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazieres, D., Kaashoek, F., Morris, R.: Labels and event processes in the asbestos operating system. In: SOSP, vol. 5, pp. 17–30 (2005)
11. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.: Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: OSDI, vol. 10, pp. 255–270 (2010)
12. Ene, A., Horne, W., Milosavljevic, N., Rao, P., Schreiber, R., Tarjan, R.E.: Fast exact and heuristic methods for role minimization problems. In: SACMAT, pp. 1–10 (2008)
13. Ferrara, A., Fuchsbauer, G., Warinschi, B.: Cryptographically enforced RBAC. In: CSF, pp. 115–129, June 2013
14. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, New York (2006)
15. Gong, L., Qian, X.: The complexity and composability of secure interoperation. In: 1994 IEEE Computer Society Symposium on Research in Security and Privacy 1994, Proceedings, pp. 190–200. IEEE (1994)
16. Haldar, V., Chandra, D., Franz, M.: Dynamic taint propagation for Java. In: ACSAC, pp. 303–311 (2005)
17. Jaume, M., Tong, V.V.T., Mé, L.: Flow based interpretation of access control: detection of illegal information flows. In: ICISS, pp. 72–86 (2011)
18. Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: ACM SIGOPS Operating Systems Review, vol. 41, pp. 321–334. ACM (2007)
19. Lam, L.C., Chiueh, T.: A general dynamic information flow tracking framework for security applications. In: ACSAC, pp. 463–472 (2006)
20. Lampson, B.W.: A note on the confinement problem. Commun. ACM **16**(10), 613–615 (1973)
21. Mao, Z., Li, N., Chen, H., Jiang, X.: Trojan horse resistant discretionary access control. In: SACMAT, pp. 237–246. ACM (2009)

22. Mao, Z., Li, N., Chen, H., Jiang, X.: Combining discretionary policy with mandatory information flow in operating systems. ACM TISSEC **14**(3), 24:1–24:27 (2011)
23. Marinovic, S., Craven, R., Ma, J., Dulay, N.: Rumpole: a flexible break-glass access control model. In: SACMAT, pp. 73–82. ACM (2011)
24. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: SIGOPS Operating Systems Review, vol. 31, pp. 129–142. ACM (1997)
25. Petritsch, H.: Break-Glass: Handling Exceptional Situations in Access Control. Springer, Heidelberg (2014)
26. Sze, W.K., Mital, B., Sekar, R.: Towards more usable information flow policies for contemporary operating systems. In: SACMAT (2014)
27. Sze, W.K., Sekar, R.: Provenance-based integrity protection for windows. In: ACSAC 2015, New York, NY, USA, pp. 211–220. ACM, New York (2015)
28. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V. (eds.) STOC, pp. 253–264. ACM (1978)
29. Yannakakis, M.: Edge-deletion problems. SIAM J. Comput. **10**(2), 297–309 (1981)
30. Zhang, D., Ramamohanrao, K., Ebringer, T.: Role engineering using graph optimisation. In: SACMAT, pp. 139–144 (2007)
31. Zhu, Y., Jung, J., Song, D., Kohno, T., Wetherall, D.: Privacy scope: a precise information flow tracking system for finding application leaks. Ph.D. thesis, UC, Berkeley (2009)
32. Zimmermann, J., Mé, L., Bidan, C.: An improved reference flow control model for policy-based intrusion detection. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 291–308. Springer, Heidelberg (2003). doi:10.1007/978-3-540-39650-5_17