The Foundational work of Harrison-Ruzzo-Ullman Revisited

Mahesh V. Tripunitara Ninghui Li
Department of Computer Sciences and CERIAS,
Purdue University, West Lafayette, IN, USA 47907.
{tripunit,ninghui}@cerias.purdue.edu

Abstract

The work by Harrison, Ruzzo and Ullman (the HRU paper) on safety in the context of the access matrix model is widely considered to be foundational work in access control. In this paper, we address two errors we have discovered in the HRU paper. To our knowledge, these errors have not been previously reported in the literature. The first error regards a proof that shows that safety analysis for mono-operational HRU systems is in **NP**. The error stems from a faulty assumption that such systems are monotonic for the purpose of safety analysis. We present a corrected proof in this paper. The second error regards a mapping from one version of the safety problem to another that is presented in the HRU paper. We demonstrate that the mapping is not a reduction, and present a reduction that enables us to infer that the second version of safety introduced in the HRU paper is also undecidable for the HRU scheme. These errors lead us to ask whether the notion of safety as defined in the HRU paper is meaningful. We introduce other notions of safety that we argue have more intuitive appeal, and present the corresponding safety analysis results for the HRU scheme.

1 Introduction

Access control enables controlled sharing of resources (henceforth called *objects*) among principals (henceforth called *subjects*). It is one of the most important areas of research in computer security and has been called the "traditional center of gravity of computer security" [5]. The work by Harrison, Ruzzo and Ullman [14] (henceforth called *the HRU paper*) on the access matrix model [12, 18] is generally considered to be foundational work in access control. The results from the paper are included in popular textbooks in computer security [4, 5, 6, 9, 10, 26], and the paper is considered required reading for researchers in access control. Since its publication, the HRU paper has had considerable impact on research in access control. At the time of submission, according to Google Scholar [11], the HRU paper had been cited 264 times in the literature.

The HRU paper presents a rather general access control scheme (henceforth called *the HRU scheme*) and introduces *safety analysis* in the context of the scheme. In the HRU scheme, an access control system is perceived as a state-change system that consists of a start-state and a state-change rule. A state is an access matrix [18], whose rows are subjects and columns are objects. A state-change rule is a set of commands, each of which consists of a conjunction of conditions and a sequence of primitive operations. Each condition tests for the presence of a right in a cell of the access matrix, and each operation is one of six types (see Section 2). In the HRU paper, safety is defined with respect to whether a *leak* can occur. Informally, a leak is the execution of an operation that causes a right to be entered into a cell in the access matrix where it does not already exist. A system is considered to be safe if there exists no sequence of state-changes that leads to a leak. In this paper, we call this version of safety (r)-leak-safety (see Table 1). Since its introduction in the

(r)-leak-safety: whether a particular right may leak.	(o,r)-leak-safety: whether a particular right may leak to a cell associated with a particular object.	(s,o,r)-leak-safety: whether a particular right may leak to the cell associated with a particular subject and object.
(r)-simple-safety: whether a particular right may appear in a cell, where it does not exist in the start-state.	(o,r)-simple-safety: whether a particular right may appear in a cell associated with a particular object, where it does not exist in the start-state.	(s,o,r)-simple-safety: whether a particular right may appear in the cell associated with a particular subject and object, where it does not exist in the start-state.

Table 1: The six versions of safety we consider in this paper. These versions either appear in the literature on safety analysis, or are natural extensions.

HRU paper, safety analysis has been considered to be a fundamental problem in access control, and there has been considerable work on safety in various contexts related to security [1, 2, 7, 15, 20, 21, 22, 23, 24, 27, 28, 29, 30, 31, 32, 33].

Apart from (r)-leak-safety, the HRU paper introduces what we call (o,r)-leak-safety (see Table 1). There are three major results in the HRU paper. The first is that (r)-leak-safety is undecidable for the HRU scheme. The second is that (o,r)-leak-unsafety can be "simulated" by (r)-leak-unsafety. (An unsafety problem is the complement of a corresponding safety problem.) The third is that (r)-leak-unsafety for mono-operational HRU systems is $\mathbf{NP}$-complete (a mono-operational HRU system is one in which there is only one primitive operation in each command).

In this paper, we first address two errors we have discovered in the HRU paper. Our discussions are constructive; for each error, we propose corrections. The first error we discuss is in the proof that shows that the problem of determining whether a mono-operational HRU system is (r)-leak-unsafe, is in $\mathbf{NP}$. We point out that the erroneous proof from the HRU paper is reproduced in several popular textbooks in computer security [6, 9, 26]. The error in the HRU paper stems from the assumption that mono-operational HRU systems are *monotonic* from the standpoint of (r)-leak-safety analysis. A monotonic system is one in which rights cannot be deleted, and subjects and objects cannot be destroyed. We demonstrate that this assumption is flawed. We then assert that the problem is indeed in $\mathbf{NP}$, and present a corrected proof.

The second error in the HRU paper that we address regards a mapping from (o,r)-leak-unsafety to (r)-leak-unsafety that is presented to demonstrate that (o,r)-leak-unsafety can be "simulated" by (r)-leak-unsafety. The HRU paper presents a brief argument in support of the validity of the mapping. It then asserts that the problem of determining whether the (r)-leak-safety instance produced by the mapping is false is equivalent to the problem of determining whether the (o,r)-leak-safety instance that is the input to the mapping is false. We identify five flaws in the argument in the HRU paper; the mapping presented in the HRU paper may lead us to make an erroneous inference that a particular (o,r)-leak-safety instance is true when its corresponding (r)-leak-safety instance is false, and vice versa. We observe that what Harrison et al. [14] intended was to demonstrate that (o,r)-leak-unsafety *reduces* [8, 16, 17, 25] to (r)-leak-unsafety. Our discovery of the flaws in the argument presented in the HRU paper demonstrates that the mapping that is presented there is not a reduction. We argue that even if a reduction does exist from (o,r)-leak-unsafety to (r)-leak-unsafety, given the assertion in the HRU paper that (r)-leak-unsafety is undecidable for the HRU scheme, the reduction would tell us nothing about the decidability or computational complexity of (o,r)-leak-unsafety. We argue, however, that the existence of a reduction from (r)-leak-unsafety to (o,r)-leak-unsafety would allow us to infer that (o,r)-leak-unsafety is also undecidable for the HRU scheme. We present exactly

such a reduction in Section 4. The particular kind of reduction we use is a polynomial-time truth table reduction [17, 25].

The two errors we have identified in the HRU paper lead us to ask whether the way safety is defined in the HRU paper is meaningful. Indeed, we ask whether (r)-leak-safety and (o,r)-leak-safety are what Harrison et al. [14] intended at all. The informal characterization for unsafety from the HRU paper is: "*. . . whether, given some initial access matrix, there is some sequence of commands in which a particular generic right is entered in some place in the matrix where it did not exist before.*" We get different versions of safety depending on how we interpret the word "before". The versions adopted in the HRU paper, which we call (r)-leak-safety and (o,r)-leak-safety, are based on the interpretation that "before" refers to the state that immediately precedes a leak.

Based on the interpretation of "before" as "in the start-state", we introduce a version of safety that we call (r)-simple-safety (see Table 1). A close examination of the HRU paper suggests that what we call (r)-simple-safety is most likely what was intended by Harrison et al. [14]. There are other justifications for why (r)-simple-safety has more intuitive appeal than (r)-leak-safety, (see Section 5). Literature on safety analysis subsequent to the HRU paper [1, 3, 19, 20, 21, 27, 30, 31, 32, 33] uses versions of safety that are different from (r)-leak-safety. In this paper, apart from (r)-leak-safety and (o,r)-leak-safety, we consider the following versions of safety that either have appeared in the literature, or are natural extensions: (s,o,r)-leak-safety, (r)-simple-safety, (o,r)-simple-safety and (s,o,r)-simple-safety. We give an informal characterization of each in Table 1, and formal definitions in Section 5. We point out that some literature on safety analysis subsequent to the HRU paper [1, 3, 27, 30, 32, 33] asserts that (r)-simple-safety and (s,o,r)-simple-safety are undecidable for the HRU scheme. To our knowledge, neither result has been shown previously. We establish these results in this paper (see Section 5). We discuss the undecidability proof from the HRU paper and point out that it can be used to show that (r)-simple-safety is undecidable for the HRU scheme. We also observe that (r)-simple-safety reduces to (o,r)-simple-safety, and that (o,r)-simple-safety reduces to (s,o,r)-simple-safety, thereby proving that both (o,r)-simple-safety and (s,o,r)-simple-safety are also undecidable for the HRU scheme.

Related work This paper primarily addresses the work of Harrison et al. [14] on safety analysis in the context of the access matrix model. Harrison and Ruzzo [13] address monotonic HRU systems. Subsequent to the work by Harrison et al. [14], there has been considerable work on safety in various contexts related to security [1, 2, 7, 15, 20, 21, 22, 23, 24, 27, 28, 29, 30, 31, 32, 33]. In particular, Sandhu [27, 30] uses what we call (r)-simple-safety as the notion of safety, while Ammann and Sandhu [1, 3], Soshi et al. [32, 33] and Solworth and Sloan [31] use what we call (s,o,r)-simple-safety. Li et al. [19, 20, 21] have generalized safety analysis to security analysis, and use the term simple safety to refer to what we call (s,o,r)-simple-safety, and bounded safety to refer to a slightly modified version of what we call (o,r)-simple-safety.

Layout The remainder of the paper is organized as follows. In the next section, we describe the HRU scheme and reproduce the definitions for a leak and for (r)-leak-safety from the HRU paper. We discuss the first error in Section 3 and the second error in Section 4. We present our notions of simple-safety and undecidability results for them in Section 5. We conclude with Section 6.

2 The HRU Scheme and Safety

An access control system is identified by a start (or current) state, γ, and a state-change rule, ψ. An access control system is based on an access control scheme, which specifies a set of states, Γ, and a set of state-change rules, Ψ. In a system based on the HRU scheme, each state $\gamma \in \Gamma$ is associated with the tuple

$\langle S_\gamma, O_\gamma, M_\gamma[\,]\rangle$, where S_γ and O_γ are finite sets of subjects and objects, respectively, that exist in γ, and $M_\gamma[\,]$: $S_\gamma \times O_\gamma \rightarrow 2^{R_\psi}$ is an access matrix that maps a $\langle subject, object\rangle$ pair to a set of rights. R_ψ is the finite set of all such rights in the system, and we use 2^{R_ψ} to denote the powerset of R_ψ. We prefer to associate the finite set of rights, R_ψ, with the state-change rule ψ rather than the state γ as the set is fixed across all states for a system. We postulate that $S_\gamma \subset \mathcal{S}$ and $O_\gamma \subset \mathcal{O}$ are countable sets of all possible subjects and objects respectively that can exist in a state. Every subject is also an object, i.e., $\mathcal{S} \subseteq \mathcal{O}$ and $S_\gamma \subseteq O_\gamma$ in every state γ.

A state-change in an HRU system is the successful execution of a command. The state-change rule ψ is specified by a finite set of rights, R_ψ, and a finite set of commands, where each command is of the following form.

$commandName(x_1, \ldots, x_a)$
    if $r_1 \in M[x_{i_1}, x_{j_1}] \wedge \cdots \wedge r_b \in M[x_{i_b}, x_{j_b}]$ then
        primitive-operation$_1$
                $\vdots$
        primitive-operation$_c$

In the above command, a and c are positive integers, and b is a non-negative integer. The string $commandName$ is the name of the command, and $x_1, \ldots, x_a$ is a list of parameters. The "if ... then" portion checks for the presence of rights in cells in the (current instance of the) access matrix, and each check is of the form $r_k \in M[x_{i_k}, x_{j_k}]$ and is called a condition. When $b = 0$, we have a command with no conditions, when $b \leq 1$, we have a mono-conditional command, and when $b \leq 2$, we have a bi-conditional command. When $c = 1$, we have a mono-operational command. A system that consists of only mono-conditional commands is called a mono-conditional system, one that consists only of bi-conditional commands is called a bi-conditional system, and one that consists of only mono-operational commands is called a mono-operational system. The safety problem in mono-conditional, bi-conditional and mono-operational HRU systems have been studied in the literature [13, 14]. We require that the parameters $x_{i_1}, \ldots, x_{i_b}$ be instantiated with subjects (otherwise, an attempt at executing the command fails). We also require that $\{r_1, \ldots, r_b\} \subseteq R_\psi$ (the r_i's do not have to be distinct from one another). If we assume that γ is the state in which we attempt to execute the command, then the "if ... then" check succeeds if and only if the rights are present in the corresponding cells of the access matrix in the state γ.

In the execution of a command, the "if ... then" conditions are first evaluated. If any of them is not met, the command fails to execute. Otherwise, the primitive operations are executed in sequence. If the execution of a primitive operation succeeds, then the access matrix is altered in a corresponding way, as we discuss below for each kind of primitive operation. If the execution of a primitive operation fails, the access matrix is unaltered. Each primitive operation is of one of the following forms. In each of the following, let $M_\gamma[\,]$ be the access matrix, S_γ the set of subjects and O_γ the set of objects that exist in the state immediately before the execution of the primitive operation, and $M_{\gamma'}[\,]$, $S_{\gamma'}$ and $O_{\gamma'}$ be the access matrix, set of subjects and set of objects, respectively, immediately after the successful execution of the primitive operation (if the execution is unsuccessful, $M_{\gamma'}[\,] = M_\gamma[\,]$, $S_{\gamma'} = S_\gamma$ and $O_{\gamma'} = O_\gamma$).

enter r *into* $M[x, y]$: when x is instantiated with s and y with o, this operation succeeds if and only if $s \in S_\gamma$ and $o \in O_\gamma$. The result is $M_{\gamma'}[s, o] = M_\gamma[s, o] \cup \{r\}$.

delete r *from* $M[x, y]$: when x is instantiated with s and y with o, this operation succeeds if and only if $s \in S_\gamma$ and $o \in O_\gamma$. The result is $M_{\gamma'}[s, o] = M_\gamma[s, o] - \{r\}$.

create subject x : when x is instantiated with s, this operation succeeds if and only if $s \in \mathcal{S} - S_\gamma$. $S_{\gamma'} = S_\gamma \cup \{s\}$, and $O_{\gamma'} = O_\gamma \cup \{s\}$. The result is that $M_{\gamma'}[\,]$ has a row and column associated with s, and $M_{\gamma'}[s, o] = \emptyset$ for all $o \in O_{\gamma'}$.

$R_\psi = \{\mathsf{own}, \mathsf{read}\}$ $S_\gamma = \{\text{alice}\}$ $O_\gamma = S_\gamma \cup \{\text{myFile}\}$

$M_\gamma[\text{alice, alice}] = \emptyset$ $M_\gamma[\text{alice, myFile}] = \{\mathsf{own}\}$

`createSubject`(x, x')
create subject x'
enter own *into* $M[x, x']$

`destroySubject`(x, x')
if own $\in M[x, x']$
destroy subject x'

`createObject`(x, y)
create object y
enter own *into* $M[x, y]$

`destroyObject`(x, y)
if own $\in M[x, y]$
destroy object y

`transferOwn`(x, x', y)
if own $\in M[x, y]$ *then*
enter own *into* $M[x', y]$
remove own *from* $M[x, y]$

`grantRead`(x, x', y)
if own $\in M[x, y] \wedge$ read $\in M[x, y]$ *then*
enter read *into* $M[x', y]$

Figure 1: An HRU system, whose current state is γ. The only subject that exists in the current state is alice, and the only object that exists in the current state, apart from alice, is myFile. The state-change rule, ψ, consists of the six commands `createSubject`, `destroySubject`, `createObject`, `destroyObject`, `transferOwn` and `grantRead`.

create object y : when y is instantiated with o, this operation succeeds if and only if $o \in \mathcal{O} - O_\gamma$. $O_{\gamma'} = O_\gamma \cup \{o\}$. The result is that $M_{\gamma'}[\,]$ has a column associated with o, and $M_{\gamma'}[s, o] = \emptyset$ for all $s \in S_{\gamma'}$.

destroy subject x : when x is instantiated with s, this operation succeeds if and only if $s \in S_\gamma$. $S_{\gamma'} = S_\gamma - \{s\}$, and $O_{\gamma'} = O_\gamma - \{s\}$. The result is that $M_{\gamma'}[\,]$ has no row or column associated with s.

destroy object y : when y is instantiated with o, this operation succeeds if and only if $o \in O_\gamma$. The result is that $O'_\gamma = O_\gamma - \{o\}$, and $M_{\gamma'}[\,]$ has no column associated with o.

Figure 1 presents an example of an HRU system. The subject alice is the only subject that exists in the state γ. The object myFile is the only object, other than alice, that exists in γ. There are two rights associated with the system, own and read. The state-change rule, ψ, consists of the six commands, `createSubject`, `destroySubject`, `createObject`, `destroyObject`, `transferOwn` and `grantRead`.

Before we discuss safety analysis, we introduce some notation regarding state-changes. When a state γ_1 can be reached from a state γ by the successful execution of the command α, we denote this as $\gamma \mapsto_\alpha \gamma_1$. When necessary, we explicitly indicate the values with which each parameter to the command is to be instantiated; for example, $\gamma \mapsto_{\alpha(s_1,s_2,o_1)} \gamma_1$. We simply write $\gamma \mapsto \gamma_1$ if the command whose execution that can cause the state-change is either irrelevant, or clear from the context. Finally, we write $\gamma \overset{*}{\mapsto}_\psi \gamma_1$, when some sequence of successful executions of commands from the state-change rule ψ can cause the state γ_1 to be reached from γ. For example, for the system shown in Figure 1, consider the state γ_1, in which $S_{\gamma_1} = \{\text{alice}\}$, $O_{\gamma_1} = S_{\gamma_1}$, and $M_{\gamma_1}[\text{alice, alice}] = \emptyset$. Then $\gamma \mapsto_{\texttt{destroyObject}} \gamma_1$, and hence, $\gamma \overset{*}{\mapsto}_\psi \gamma_1$.

Given an HRU system, we can easily check whether a subject has a right to an object in the current state by consulting the relevant cell in the access matrix. However, we may also want to ask questions regarding states that are reachable from the current state. The HRU paper presents basic versions of such verification problems for the HRU scheme. It calls such a problem safety analysis. A system is considered to be safe for a given right if that right cannot be *leaked* to a cell, that is, be entered into a cell where it does not exist in the current state. We now reproduce a definition for safety that is presented in the HRU paper. As the definition for safety utilizes the notion of the leakage of a right, we first define what it means for the execution of a

command to leak a right. We call this version of safety (r)-leak-safety. The "(r)" stands for "with respect to a right", as we fix a right and ask the safety question. The "leak" indicates that this is a definition based on the notion of a leak. As we shall see, there are other versions of safety that arise as we continue our discussions in this paper.

Definition 1 (Leak) Given a command α in an HRU system and the current state γ of the system, we say that α leaks the right r from γ if in the execution of α for some instantiation of its parameters in the state γ, the execution of a primitive operation (presumably of the form *enter* r *into* $M[s, o]$) in α enters r into a cell of the access matrix which did not contain r immediately before the execution of the primitive operation.

Definition 2 ((r)-leak-safety) Given an HRU system with the start-start γ, a state-change rule ψ, and a right $r \in R_\psi$, we say that the system is (r)-leak-unsafe for r if and only if there is a state γ_n and a command α in the state-change rule ψ for the system such that:

- $\gamma \overset{*}{\mapsto}_\psi \gamma_n$, and,
- α leaks r from γ_n.

Consider the HRU system shown in Figure 1. The command $createObject$ leaks the right own from the state γ, as the primitive operation that adds own to a cell in which it did not previous exist can be successfully executed by the execution caused by the invocation $createObject$ (alice, newFile). Therefore, the state γ and the system are (r)-leak-unsafe for the right own. However, the system is (r)-leak-safe for the right read, as there exists no state that is reachable from γ from which read can be leaked by any command in ψ. The reason is that the only command that can leak read is `grantRead`, which requires that a subject already possess the read right. No subject possesses the right in the state γ, and therefore no subject can possess the right in any state that is reachable from γ.

3 The NP Proof for Mono-Operational HRU Systems

Given the definition for the (r)-leak-safety problem from the HRU paper that we reproduce in the previous section, the HRU paper considers the question of how efficiently one can determine whether a mono-operational HRU system is (r)-leak-unsafe. It is asserted in the HRU paper that, for such systems, the problem is **NP**-complete. We have discovered an error in the proof presented in the HRU paper, that shows that the problem is in **NP**. We first discuss the error, and then its implications. The problem is indeed in **NP** as claimed, and we provide a corrected proof in this section.

The incorrect proof proceeds as follows [14].

> *The proof hinges on two simple observations. First, commands can test for the presence of rights, but not the absence of rights or objects. This allows delete and destroy commands to be removed from computations leading to a leak (since the system is mono-operational, we can identify the command by the type of primitive operation). Second, a command can only identify objects by the rights in their row and column of the access matrix. No mono-operational command can both create an object and enter rights, so multiple creates can be removed from computations, leaving the creation of only one subject. This allows the length of the shortest 'leaky' computation to be bounded.*

Harrison et al. [14] use the phrase "shortest leaky computation" to refer to the shortest sequence of state-changes that results in a state from which there is a command that leaks the right, and we use this phrase as well in the remainder of this section. The proof proceeds to show, based on the two observations made above,

that an upper-bound on the length of the shortest leaky computation is $|R_\psi|\,(|S_\gamma|+1)\,(|O_\gamma|+1)+1$, where γ is the start-state and ψ is the state-change rule, which is polynomial in the size of the system. Therefore, if the system is (r)-leak-unsafe, then there is an evidence whose size is polynomially-bounded, and that can be verified in polynomial-time, thereby demonstrating that the problem is in $\mathbf{NP}$.

The error in the proof in the HRU paper is with the first observation reproduced above. What the first observation asserts is that mono-operational HRU systems are monotonic from the standpoint of (r)-leak-safety analysis. That is, *delete* and *destroy* commands are inconsequential from the standpoint of determining whether a particular mono-operational HRU system is (r)-leak-safe or not, and therefore can be ignored. While monotonic systems seem to have little practical utility, the observation that realistic (non-monotonic) systems can be "approximated" by monotonic systems from the standpoint of safety analysis has since been used to establish the decidability of safety in various contexts (for example, by Ammann and Sandhu [1]).

We assert that we cannot ignore *delete* or *destroy* commands in the (r)-leak-safety analysis of mono-operational HRU systems. The reason is that the definition for (r)-leak-unsafety asks only whether there is a state-change sequence in which the final state-change is a leak (we do not have to consider "partial" executions of commands as our focus here is on mono-operational systems). Consequently, a system in which a subject has a right in the start-state, loses that right (owing to the execution of a *delete* or *destroy* command) and re-acquires that right is a system that is (r)-leak-unsafe. For example, consider a system whose start-state is γ and state-change rule is ψ, and $S_\gamma = \{s\}$, $O_\gamma = S_\gamma$, $R_\psi = \{r\}$, $M_\gamma[s,s] = \{r\}$, and ψ consists of the following two commands.

$enterRight(p)$	$removeRight(p)$
enter r *into* $M[p,p]$	*delete* r *from* $M[p,p]$

The above mono-operational HRU system is (r)-leak-unsafe for the right r. The following is the shortest leaky computation: $\gamma \mapsto_{removeRight(s)} \gamma_1$. This results in $M_{\gamma_1}[s,s] = \emptyset$. From γ_1, the command $enterRight$ leaks r. We cannot remove the execution of $removeRight$ from the above leaky computation (thereby resulting in a shortest leaky computation of length 0) as then, there is no command that leaks r from γ. The example demonstrates that we cannot ignore *delete* and *destroy* commands in considering the shortest leaky computation in a mono-operational HRU system.

We now present the corrected proof to show that deciding whether a mono-operational HRU system is (r)-leak-unsafe is in $\mathbf{NP}$. The correction we have incorporated to our proof is to deal with the cases that a delete or destroy command is part of the shortest leaky computation.

Theorem 1 The problem of determining whether a mono-operational HRU system is (r)-leak-unsafe is in $\mathbf{NP}$.

Proof. We seek to show that if a mono-operational HRU system is (r)-leak-unsafe for a given right r, then there exists an evidence whose size is polynomially-bounded in the size of the components of the system (given the start-state γ and state-change rule ψ, the components of the system are $S_\gamma, O_\gamma, M_\gamma[\,]$ and R_ψ), and that this evidence can be verified in polynomial time. The evidence is the shortest leaky computation, and as in the proof provided in the HRU paper, we provide an upper-bound for its length. As the system is mono-operational, each command is identified by the primitive operation in it.

Let $C = \gamma_0 \mapsto \cdots \mapsto \gamma_n$, where $\gamma_0 = \gamma$, be the shortest leaky computation, and $|C|$ denote its length. Assume that an *enter* command leaks the right r into the cell $M[s_l, o_l]$ for $s_l \in S_{\gamma_n}$ and $o_l \in O_{\gamma_n}$. We look at the first *create* command in C. There are the following cases.

1. Such a command does not exist. Then C does not have any *destroy* commands either, as any such commands can be removed without affecting the leak. Observe that the subjects/objects being destroyed are not re-created in C. Furthermore, there can be at most one *delete* command, which is "*delete* r from $M[s_l, o_l]$". Any other *delete* command can be removed from the sequence without affecting the leak. Consequently, in this case, $|C| \leq |R_\psi| \left(|S_{\gamma_0}| + 1\right) \left(|O_{\gamma_0}| + 1\right) + 1$.
2. The command exists and is *create subject* s, and s is not in S_γ. Then C does not have any other *destroy* or *create* commands. Otherwise, one can remove all these commands and replace each subject created later with s, and the resulting sequence would be a shorter leaky computation. Observe that if s_l (or o_l) is destroyed and re-created, then the new incarnation of s_1 (or o_1) is no different from s, and a leak would still occur. Furthermore, there can be at most one *delete* command, which is "*delete* r from $M[s_l, o_l]$". Any other *delete* command can be removed from the sequence without affecting the leak. Consequently, $|C| \leq |R_\psi| \left(|S_{\gamma_0}| + 1\right) \left(|O_{\gamma_0}| + 1\right) + 2$.
3. The command exists and is *create subject* s, and $s \in S_\gamma$. Then there is one *destroy subject* s command before this command. Furthermore, s must be one of s_l or o_l; otherwise, removing the *destroy* and the *create* commands gives us a shorter leaky computation. Also C does not have any other *destroy* or *create* commands. Otherwise, we can remove all these commands, and replace each subject created later with s, and the resulting sequence would be a shorter leaky computation. Finally, there is no *delete* command. Observe that a "*delete* r from $M[s_l, o_l]$" before the create command is not needed, as all rights related to s are cleared when it is destroyed and re-created. Observe also that a "*delete* r from $M[s_l, o_l]$" after the create command is not needed either, as the right r must not exist in $M[s_l, o_l]$ at that point; otherwise, a leak has already occurred before the *delete* command. Consequently, in this case, $|C| \leq |R_\psi| \left(|S_{\gamma_0}| + 1\right) \left(|O_{\gamma_0}| + 1\right) + 2 + |R_\psi| \left(|S_{\gamma_0}| + |O_{\gamma_0}| + 1\right)$.
4. The command exists and is *create object* o, and o is not in O_γ. Then C does not have any other *destroy object* or *create object* commands, as these can be removed to give a shorter leaky computation. (Note that we cannot replace this command with a *create subject* command, as such a command may not exist.) We then look for the next *create* command. One of the preceding three cases would apply, as if it is found, it must be a *create subject*. Therefore, $|C| \leq |R_\psi| \left(|S_{\gamma_0}| + 1\right) \left(|O_{\gamma_0}| + 2\right) + 3 + |R_\psi| \left(|S_{\gamma_0}| + |O_{\gamma_0}| + 2\right)$.
5. The command exists and is *create object* o, and $o \in O_\gamma$. Then there is one *destroy object* o command before this command. Furthermore, o must be o_l; otherwise, removing the *destroy* and the *create* commands, we have a shorter leaky computation. Also C does not have any other *destroy object* or *create object* commands. Otherwise, one can remove all these commands, and replace each object created later with o, the resulting sequence would be a shorter leaky computation. We then look for the next *create* command. One of the preceding three cases would apply, as if it is found, it must be a *create subject*. Therefore, in this case, $|C| \leq |R_\psi| \left(|S_{\gamma_0}| + 1\right) \left(|O_{\gamma_0}| + 2\right) + 4 + |R_\psi| \left(2|S_{\gamma_0}| + |O_{\gamma_0}| + 2\right)$.

We have demonstrated that if a mono-operational HRU system is (r)-leak-unsafe, then there exists an evidence (the shortest leaky computation) whose size is polynomially bounded, and that can be verified in polynomial time. ∎

4 The Reduction from One Safety Problem to Another

As we mention in Section 1, the HRU paper discusses two versions of safety: (r)-leak-safety and (o,r)-leak-safety. In this section, we first define (o,r)-leak-safety. In the HRU paper, (o,r)-leak-safety is characterized

only informally as: "*Another common notion of the term 'safety' is that one be assured it is impossible to leak right r to a particular object o_1.*" Consequently, our definition for (o,r)-leak-safety is similar to the definition for (r)-leak-safety, but captures the above intuition intended by Harrison et al. [14]. After presenting the definition for (o,r)-leak-safety, we reproduce a mapping from (o,r)-leak-unsafety to (r)-leak-unsafety that is presented in the HRU paper. In presenting the mapping, the HRU paper asserts: "*We can use our more general definition of safety to simulate this one.*" The "general definition of safety" is (r)-leak-safety, and "this one" is (o,r)-leak-safety. The second error we have discovered in the HRU paper regards this mapping, and we discuss this and related issues in this section.

The HRU paper does not precisely characterize the properties the mapping presented there possesses. We argue that as safety is what is emphasized, it is reasonable to expect that the mapping must be a *reduction* [8, 16, 17, 25]. A reduction from a decision problem A to a decision problem B is a computable mapping that enables us to decide an instance of A provided that we can decide corresponding instances of B. This is precisely what the HRU paper seeks; the ability to answer (o,r)-leak-safety instances for a system provided that (r)-leak-safety instances can be answered for that system.

After reproducing the mapping from (o,r)-leak-unsafety to (r)-leak-unsafety that is presented in the HRU paper, we present five flaws we have identified in the mapping that preclude it from being a reduction. Therefore, the mapping cannot be used to infer any meaningful relationship between (r)-leak-safety and (o,r)-leak-safety.

Definition 3 ((o,r)-leak-safety) Given an HRU system with the start-state γ, a state-change rule ψ, a right $r \in R_\psi$, and an object $o \in \mathcal{O}$, we say that the system is (o,r)-leak-unsafe for $\langle o, r\rangle$ if and only if there exists a state γ_n and a command α in ψ, such that both of the following are true.

- $\gamma \overset{*}{\mapsto}_\psi \gamma_n$, and,
- α leaks r from γ_n into one of the cells in the column corresponding to o in the access matrix.

Given the above definition, we are able to immediately establish the following two lemmas. The first lemma relates (r)-leak-safety and (o,r)-leak-safety via implication. The second establishes that (o,r)-leak-safety for two objects that do not exist in the start-state are equivalent. These results are used in Section 4.2 to demonstrate that (r)-leak-safety reduces to (o,r)-leak-safety.

Lemma 2 Given an HRU system $\langle \gamma, \psi\rangle$ where γ is the start-state, ψ is the state-change rule and $r \in R_\psi$ is a right. If the system is (r)-leak-unsafe for r, then there exists $o \in \mathcal{O}$ such that the system is (o,r)-leak-unsafe for $\langle o, r\rangle$.

Lemma 3 Given an HRU system with the start-state γ, the state-change rule ψ, a right $r_1 \in R_\psi$, and the subjects s_1 and s_2 such that $s_1 \neq s_2$, and $\{s_1, s_2\} \subseteq \mathcal{S} - S_\gamma$. The system is (o,r)-leak-safe for $\langle s_1, r_1\rangle$ if and only if it is (o,r)-leak-safe for $\langle s_2, r_1\rangle$.

The proofs for both the above lemmas is straightforward. For Lemma 2, if the system is (r)-leak-unsafe for r, then there is a state-change sequence that leads to a command that leaks r. Let the leaky command leak r to the column corresponding to the object o. Then, the system is (o,r)-leak-unsafe for $\langle o, r\rangle$. For Lemma 3, we recall that every subject is also an object. If the system is (o,r)-leak-unsafe for $\langle s_1, r_1\rangle$, then there is a corresponding state-change sequence that leads to a command that leaks r_1 to a cell in the column corresponding to s_1. In the state-change sequence, we replace every occurrence of s_1 with s_2, and this gives us a state-change sequence that leads to a command that leaks r_1 to a cell in the column corresponding to s_2.

4.1 The mapping from one safety problem to another from the HRU paper

The HRU paper presents the following mapping from (o,r)-leak-unsafety to (r)-leak-unsafety that we call M_{HRU}. Given a system with start-state γ and state-change rule ψ, we consider the (o,r)-leak-unsafety problem for $\langle o_1, r_1 \rangle$. M_{HRU} produces as output a state γ', a state-change rules ψ' and a right r'_1 that is the specification for the corresponding (r)-leak-unsafety problem. The state-change rule ψ' is such that $R_{\psi'} = R_\psi \cup \{r'_1, r'_2\}$, where $\{r'_1, r'_2\} \cap R_\psi = \emptyset$. Every command that is in ψ is also in ψ'. In addition, ψ' has the following command.

$$
\begin{array}{l}
dummy(s, o) \\
\quad if\ r_1 \in M[s, o] \wedge r'_2 \in M[o, o]\ then \\
\quad\quad enter\ r'_1\ into\ M[o, o]
\end{array}
$$

The start-state γ' is the same as γ, except that $r'_2 \in M_{\gamma'}[o_1, o_1]$. The HRU paper then makes the claim that "*...leaking r'_1 to anybody is equivalent to leaking r_1 to object o_1 specifically.*" We assert that the claim is false for the following reasons.

1. Under the mapping, we enter the right r'_2 into the cell $M[o_1, o_1]$ in the start-state. However, if o_1 is an object that is not a subject, then there is no such cell in the access matrix.
2. It is assumed that the object o_1 in the (o,r)-leak-safety instance already exists. The mapping does not address the case that $o_1 \in \mathcal{O}$, but does not exist in the start-state. It is possible that the system is (o,r)-leak-unsafe, as o_1 may be created by the execution of some command. A practical situation in which this case is interesting regards configuration files. Such files are often identified by name and are used by application programs such as web browsers. Furthermore, many programs create a configuration file if one does not already exist. Therefore, even if the object o_1 (a configuration file) does not exist in the start-state, we may want to ask a corresponding (o,r)-leak-safety question.
3. The case that o_1 may be destroyed is not addressed. If there is a command in the system that can destroy o_1, then even if o_1 is subsequently re-created, the right r'_2 that the mapping includes in the cell $M[o_1, o_1]$ is no longer present. Therefore, we may incorrectly infer that the system is (o,r)-leak-safe, if the (r)-leak-safety property is true for the system produced under M_{HRU}. This case is also of practical interest for reasons similar to the previous case; o_1 may be a configuration file that is destroyed and re-created.
4. The case that r_1 is leaked to some cell in the column corresponding to o_1 in a "transient state" (that is, the partial execution of a command) is not addressed. It is possible that r_1 is entered into some cell in the column corresponding to o_1, and then removed in the same command. This would make the system $\langle \gamma, \psi \rangle$ (o,r)-leak-unsafe for $\langle o_1, r_1 \rangle$. However, the conditions for the command $dummy$ would never be satisfied, and the system $\langle \gamma', \psi' \rangle$ would be (r)-leak-safe for r'_1 leading us to infer incorrectly that the system $\langle \gamma, \psi \rangle$ is (o,r)-leak-safe for $\langle o_1, r_1 \rangle$.
5. Suppose that some subject s_1 has the right r_1 to o_1 in the start-state, that r_1 can never be deleted from the cell corresponding to s_1 and o_1, and that neither s_1 nor o_1 can be destroyed. Then, we may incorrectly infer that the system is (o,r)-leak-unsafe, because the conditions for the command $dummy$ are satisfied. For the mapping to be correct, we need to assume that no subject has the right r_1 over o_1 in the start-state.

We do not attempt to directly correct these flaws in the mapping. The reason is that even if there does exist a reduction from (o,r)-leak-safety to (r)-leak-safety, we cannot infer anything meaningful about the decidability or computational complexity of (o,r)-leak-safety in the HRU scheme. As it is shown in the

```
1  M^{L_orls} (⟨γ, ψ⟩, r_1)
2    /* Inputs:
3         - An HRU system, ⟨γ, ψ⟩
4         - A right, r_1 ∈ R_ψ
5       Output:
6         - true, if the system ⟨γ, ψ⟩ is (r)-leak-safe for r_1
7         - false, otherwise */
8    For each o ∈ O_γ
9      query the oracle L_orls whether ⟨γ, ψ⟩ is (o,r)-leak-safe for ⟨o, r_1⟩
10     if false, return false
11   For some s ∈ S − S_γ
12     query the oracle L_orls whether ⟨γ, ψ⟩ is (o,r)-leak-safe for ⟨s, r_1⟩
13     if false, then return false
14   return true
```

Figure 2: $M^{L_{orls}}$ is an oracle Turing machine that decides L_{rls}. The above algorithm returns true if the system $\langle \gamma, \psi \rangle$ is (r)-leak-safe for the right r_1, and false otherwise. $M^{L_{orls}}$ issues $|O_\gamma| + 1$ non-adaptive queries to the oracle L_{orls}.

HRU paper that (r)-leak-safety is undecidable for the HRU scheme, a reduction from (o,r)-leak-safety to (r)-leak-safety tells us nothing about how efficiently we can solve (o,r)-leak-safety for the HRU scheme. Indeed, we cannot even infer whether (o,r)-leak-safety is decidable or not for the scheme. A reduction in the other direction, if one exists, would be more meaningful, as we could then infer that (o,r)-leak-safety is also undecidable for the HRU scheme. In the next section, we provide exactly such a reduction.

4.2 A reduction from (r)-leak-safety to (o,r)-leak-safety

In this section, we present a reduction from the (r)-leak-safety to the (o,r)-leak-safety for the HRU scheme. The particular kind of reduction we present is a polynomial-time truth table reduction [17, 25]. It is known [17] that if there exists such a reduction from problem A to B, then B is a upper-bound for A, within a polynomial factor, from the perspective of computational complexity. Of course, if A is undecidable, then so is B.

A polynomial-time truth table reduction is a special case of a polynomial-time Turing reduction, which is also called a Cook reduction [8]. These reductions are based on the notion of an oracle Turing machine [25]. An oracle Turing machine, with oracle L, is denoted as M^L. L is a language, that is, a set of strings. M^L is a two-tape deterministic Turing machine. The extra tape is called the oracle tape. M^L has three additional states: $q_?$ (the query state), and q_{yes} and q_{no} (the answer states). The computation of M^L proceeds like in any ordinary Turing machine, except for transitions from $q_?$. When M^L enters $q_?$, it checks whether the contents of the oracle tape are in L. If so, M^L moves to q_{yes}. Otherwise, M^L moves to q_{no}. In other words, M^L is given the ability to "instantaneously" determine whether a particular string is in L or not. To show that there is a polynomial-time truth table reduction from a language L' to L, we need to show that there exists an oracle Turing machine M^L that decides L', and that asks only polynomially many non-adaptive queries of the form "$y \in L$". By non-adaptive queries we mean that the choice of a query does not depend on the answer to any other query; that is, all queries are specified before any of them is answered.

In our case, we let L_{rls} be the language that contains all tuples of the form $\langle \sigma, r \rangle$, where $\sigma = \langle \gamma, \psi \rangle$ is an HRU system, $r \in R_\psi$ is a right in the system, and σ is (r)-leak-safe for the right r. Also, we let L_{orls} be the language that contains all tuples of the form $\langle \sigma, \langle o, r \rangle \rangle$ where $\sigma = \langle \gamma, \psi \rangle$ is (o,r)-leak-safe for $\langle o, r \rangle$.

Theorem 4 There exists a polynomial-time truth table reduction from L_{rls} to L_{orls}.

Proof. In Figure 2, we specify an oracle Turing machine,$M^{L_{orls}}$, as an algorithm. We now show that $M^{L_{orls}}$ a polynomial-time truth table reduction from L_{rls} to L_{orls}. We first observe that $M^{L_{orls}}$ asks $|O_\gamma|+1$ queries, which is certainly polynomial in the size of the input. That $M^{L_{orls}}$ decides L_{rls} follows from Lemmas 2 and 3. ∎

The undecidability of (o,r)-leak-safety The HRU paper presents a proof to demonstrate that (r)-leak-safety is undecidable. Given the above reduction from (r)-leak-safety to (o,r)-leak-safety, we infer that (o,r)-leak-safety is undecidable as well for the HRU scheme.

In this context, we point out a minor error in the proof for the assertion that (r)-leak-safety is undecidable that is presented in the HRU paper. In the proof, a reduction from the halting problem for Turing machines to (r)-leak-unsafety is presented; a Turing machine enters a final state q_f and halts if and only if the corresponding HRU system leaks a right r_f. It is assumed in the reduction that the Turing machine's initial state is not q_f. In the case that the Turing machine's initial state is q_f, no state-changes would occur in the corresponding HRU system, and therefore, no leak can occur. This error in the proof is easily fixed by first checking whether a given Turing machine's initial state is the state q_f. If so, we know trivially that the Turing machine halts in state q_f. Only if the Turing machine's initial state is not q_f do we encode its halting problem as an (r)-leak-unsafety problem for an HRU system.

An interesting point is that Theorem 4 applies to special cases of the HRU scheme that have been studied in the literature [13, 14] as well. That is, (r)-leak-safety for mono-conditional, bi-conditional and mono-operational HRU systems reduces to (o,r)-leak-safety for mono-conditional, bi-conditional and mono-operational HRU systems respectively. Consequently, we infer that (o,r)-leak-unsafety for mono-operational HRU systems is $\mathbf{NP}$-hard. We can in fact use a proof similar to the proof for Theorem 1 to demonstrate that the problem is also in $\mathbf{NP}$, thereby making it $\mathbf{NP}$-complete.

5 More Meaningful Definitions of Safety

Our discovery of the two errors we discuss in Sections 3 and 4 leads us to ask whether the way safety is defined in the HRU paper is meaningful. We discuss this issue in this section.

5.1 Alternate definitions for safety

In this section, we consider alternate definitions for safety. These versions of safety are not based on the notion of a leak, but on whether a right may appear where it does not exist in the start-state.

Definition 4 ((r)-simple-safety) Given an HRU system with the start-state γ, a state-change rule ψ and a right r in the system, we say that the system is (r)-simple-unsafe if and only if there is a state γ_n such that all of the following are true.

- $\gamma \overset{*}{\mapsto}_\psi \gamma_n$,
- there exist $s \in S_{\gamma_n}$ and $o \in O_{\gamma_n}$ such that $r \in M_{\gamma_n}[s, o]$, and,
- either $s \notin S_\gamma$, or $o \notin O_\gamma$ or $r \notin M_\gamma[s, o]$.

We recall from Section 1 that the HRU paper informally characterizes safety as: "*. . . whether, given some initial access matrix, there is some sequence of commands in which a particular generic right is entered in*

some place in the matrix where it did not exist before." In the above definition for (r)-simple-safety, we adopt the interpretation that "before" means "in the start-state."

There are three important differences between (r)-leak-unsafety (see Definition 2) and (r)-simple-unsafety.

1. If a system is (r)-simple-unsafe for r, then it is (r)-leak-unsafe for r, but not vice versa. Intuitively, what we mean is that for the system to reach a "bad" state (be (r)-simple-unsafe), a "bad" state-change (leak) must occur. However, not every "bad" state-change leads to a "bad" state. In particular, if a subject s has the right r over the object o in the start-state γ, and there is a sequence of state-changes by which s loses the right r to o, and then re-acquires it, the system is (r)-leak-unsafe for r. However, this does not demonstrate that the system is (r)-simple-unsafe for r.
2. In (r)-leak-unsafety, the execution of a "leaky" command may fail. All Definition 2 requires is that there is some operation in the "leaky" command that enters the right where it does not exist in the access matrix immediately preceding the execution of the operation. Even if a subsequent operation fails, we have a leak, and the system is (r)-leak-unsafe. In (r)-simple-safety, we require that every command that results in the right being entered into the cell succeed.
3. In (r)-leak-unsafety, the "leaky" command may enter the right to a cell, and then delete it. This point is made explicitly in the HRU paper: "*... note that (a command) α leaks (the right) r from (the state) Q even if α deletes r after entering it.*" The rationale provided for this is that there may be "code" in between the entering of the right and the deletion of the right that is not "directly modeled". For a system to be (r)-simple-unsafe, when the right is entered into a cell, that right must persist in the cell across a state-change (execution of a command).

Definition 5 ((o,r)-simple-safety) Given an HRU system with the start-state γ, state-change rule ψ, a right $r \in R_\psi$, and an object $o \in \mathcal{O}$, we say that the system is (o,r)-simple-unsafe for $\langle o, r \rangle$ if and only if there exists a state γ_n such that all of the following are true.

- $\gamma \overset{*}{\mapsto}_\psi \gamma_n$,
- $o \in O_{\gamma_n}$ and there exists a subject $s \in S_{\gamma_n}$ such that $r \in M_{\gamma_n}[s, o]$, and,
- either $s \notin S_\gamma$, or $o \notin O_\gamma$, or $r \notin M_\gamma[s, o]$.

Definition 6 ((s,o,r)-simple-safety) Given an HRU system with the start-state γ, a state-change rule ψ, a right $r \in R_\psi$, an object $o \in \mathcal{O}$, and a subject $s \in \mathcal{S}$, we say that the system is (s,o,r)-simple-unsafe for $\langle s, o, r \rangle$ if and only if there exists a state γ_n such that all of the following are true.

- $\gamma \overset{*}{\mapsto}_\psi \gamma_n$,
- $o \in O_{\gamma_n}$, $s \in S_{\gamma_n}$, and $r \in M_{\gamma_n}[s, o]$, and,
- either $s \notin S_\gamma$, or $o \notin O_\gamma$, or $r \notin M_\gamma[s, o]$.

5.2 Simple-safety versus leak-safety

We now argue that the notions of simple-safety have more intuitive appeal than the notions of leak-safety. Indeed, we argue that simple-safety is most likely what Harrison et al. [14] intended. In support of our argument, we make the following observations.

- The proof presented in the HRU paper for the assertion that the unsafety problem for mono-operational systems is in **NP** is correct if we adopt (r)-simple-safety instead of (r)-leak-safety. That is, from the standpoint of (r)-simple-safety analysis, mono-operational HRU systems are monotonic. The reason is that if s possesses r over o in the start-state, then an evidence that the system is (r)-simple-unsafe cannot be the addition of r to $M[s, o]$. Therefore, we can ignore *delete* and *destroy* commands in our consideration of the shortest leaky computation.
- Consider the five flaws in the mapping from (o,r)-leak-safety to (r)-leak-safety presented in the HRU paper, that we discuss in Section 4.1. The flaws (3)-(5) are irrelevant if we assume that the mapping is from (o,r)-simple-safety to (r)-simple-safety. Flaw (3) addresses deletion of the object o_1. However, we do not have to consider this case for (o,r)-simple-safety as a state-change sequence that involves a command that deletes o_1 cannot be used to demonstrate that the system is (o,r)-simple-unsafe for $\langle o_1, r\rangle$ (for some right r). For flaw (4), we know that we do not have to consider "transient" state for simple-safety. The argument for flaw (5) is similar to the argument for flaw (3).
- The notions of leak-safeties mandates a peculiar transactional behavior for commands. The HRU paper does not explicitly specify, when a command contains several primitive operations, whether these operations are executed in a transactional manner or not. One issue is that whether the state in between the execution of two operations is visible to the subjects. The fact that even if a right is entered in a transient state is considered to be a leak means that every such transient state is visible to the subjects; that is, subjects can access objects using the rights they possess in such transient states. (Otherwise, there is no reason to consider a right in a transient state leakage.) For example, a subject may be able to write something to a file. An issue then is what happens when a primitive operation fails at one of these transient states. Clearly, the system cannot stay at the transient state to allow other commands to execute. Therefore, the access control system has to roll back to the state before the command is executed. It is unclear whether the changes being made in the transient states are also rolled back or not.

  That every transient state is visible to the subjects and the command rolls back if one operation fails seems to be the transactional behavior that follows from the notion of leak-safety. We find this transactional behavior rather strange. First, we are not aware of any access control system that is actually implemented in this way. Second, there is no way to make several changes to the access matrix in an atomic way. Third, a user is able to exercise the right even though the command execution that resulted in him being granted the right failed in its execution. On the other hand, simple safety implies that each command is executed in an atomic fashion, which is a more reasonable transactional behavior.
- Literature subsequent to the HRU paper has considered only what we call simple-safety, and assume that the HRU paper addresses simple-safety. For instance, Sandhu [27, 30] considers what we call (r)-simple-safety. while Ammann and Sandhu [1, 3], Li et al. [19, 20, 21], Solworth and Sloan [31] and Soshi et al. [32, 33] consider what we call (s,o,r)-simple-safety. This further supports our argument that simple-safety is more meaningful than leak-safety.

5.3 The undecidability of (r)-simple-safety and (s,o,r)-simple-safety for the HRU scheme

Some work subsequent to the HRU paper [1, 3, 19, 20, 21, 27, 30, 31, 32, 33] adopt what we call (r)-simple-safety and (s,o,r)-simple-safety as the notion of safety, and assert that safety is undecidable for the HRU scheme. To our knowledge, these results have not been established in the literature. We do so in this section.

We first consider the undecidability of (r)-simple-safety and (s,o,r)-simple-safety for the HRU scheme. We point out that the proof in the HRU paper for the undecidability of (r)-leak-safety (with the minor correction we discuss in Section 4.2) applies to (r)-simple-safety as well. In that proof, the halting problem for Turing machines is reduced to a safety problem for the HRU scheme. The Turing machine enters the state q_f if and only if a corresponding right r_f that does not exist in any cell in the start-state is entered into some cell by the execution of commands. The proof does not use "transient" states, and the right r_f is never deleted and re-entered. Consequently, we are able to assert the following theorem.

Theorem 5 (r)-simple-safety is undecidable for the HRU scheme.

Our proof for Theorem 4 that demonstrates that (r)-leak-safety reduces to (o,r)-leak-safety can easily be adapted to show that (r)-simple-safety reduces to (o,r)-simple-safety. We let L_{orss} is the language that consists of all tuples of the form $\langle \sigma, \langle o, r \rangle \rangle$ where the system $\sigma = \langle \gamma, \psi \rangle$ is (o,r)-simple-safe for $\langle o, r \rangle$, and L_{rss} be the language that consists of all tuples of the form $\langle \sigma, r \rangle$, where σ is (r)-simple-safe for r. Now, we can build an oracle Turing machine $M^{L_{orss}}$ that decides L_{rss}. $M^{L_{orss}}$ asks $|O_\gamma| + 1$ queries of the form "$y \in L_{orss}$", just as in the proof for Theorem 4. Lemmas 2 and 3 have their counterparts in (r)-simple-safety and (o,r)-simple-safety as well. Therefore, we are able to assert the following corollary.

Corollary 6 There exists a polynomial-time truth table reduction from L_{rss} to L_{orss}.

Suppose that L_{sorss} is the language that contains all tuples of the form $\langle \sigma, \langle s, o, r \rangle \rangle$ where the system σ is (s,o,r)-simple-safe for $\langle s, o, r \rangle$. Then, the proof for Theorem 4 can be adapted to show that there exists an oracle Turing machine $M^{L_{sorss}}$ that decides L_{orss}. $M^{L_{sorss}}$ works similar to $M^{L_{orss}}$, except that it asks $|S_\gamma| + 1$ queries of the form "$z \in L_{sorss}$". As Lemmas 2 and 3 have their counterparts in (o,r)-simple-safety and (s,o,r)-simple-safety, we are able to assert the following corollary.

Corollary 7 There exists a polynomial-time truth table reduction from L_{orss} to L_{sorss}.

From Corollaries 6 and 7 we are able to assert the following theorem.

Theorem 8 (s,o,r)-simple-safety is undecidable for the HRU scheme.

6 Conclusions

We have discussed two errors we have discovered in the HRU paper. The first error is in a proof that shows that the problem of determining whether a mono-operational HRU system is (r)-leak-unsafe is in $\mathbf{NP}$. We have demonstrated that an underlying assumption that mono-operational HRU systems are monotonic from the standpoint of safety analysis is faulty, and subsequently presented a corrected proof. The second error is in an argument in support of a mapping from one version of safety to another. We have demonstrated that the mapping is not a reduction, and presented a reduction in a more meaningful direction. We have also introduced the notions of simple-safety and argued that they have more intuitive appeal than leak-safety. We have also established undecidability results for the notions of simple-safety that have been assumed in the literature.

References

[1] Paul Ammann and Ravi S. Sandhu. Safety analysis for the extended schematic protection model. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 87–97, May 1991.

[2] Paul Ammann and Ravi S. Sandhu. The extended schematic protection model. *Journal of Computer Security*, 1(3-4):335–383, 1992.

[3] Paul Ammann and Ravi S. Sandhu. One-representative safety analysis in the non-monotonic transform model. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 138–149. IEEE Computer Society Press, 1994.

[4] Edward Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall PTR, Upper Saddle River, NJ 07458, 1994.

[5] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[6] Matt Bishop. *Computer Security — Art and Science*. Addison-Wesley, 2003.

[7] T. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12(6):413–430, 1983.

[8] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd IEEE Symposium on Foundations of Computer Science*, pages 151–158. IEEE Computer Society Press, 1971.

[9] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1 edition, 1982.

[10] Dieter Gollmann. *Computer Security*. John Wiley and Sons, Inc., New York, NY, 1999.

[11] Google, Inc. Google Scholar, May 2005. http://scholar.google.com/.

[12] G. Scott Graham and Peter J. Denning. Protection — principles and practice. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 40, pages 417–429. AFIPS Press, May 16–18 1972.

[13] Michael A. Harrison and Walter L. Ruzzo. Monotonic protection systems. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 461–471. Academic Press, Inc., 1978.

[14] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[15] Anita K. Jones, Richard J. Lipton, and Lawrence Snyder. A linear time algorithm for deciding security. In *17th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 33–41, October 1976.

[16] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[17] R.E. Ladner, N.A. Lynch, and A.L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[18] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971. Reprinted in ACM Operating Systems Review, 8(1):18-24, Jan 1974.

[19] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management, 2004. Accepted to appear in *Journal of the ACM*.

[20] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 126–135, June 2004.

[21] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 123–139. IEEE Computer Society Press, May 2003.

[22] Richard J. Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[23] Naftaly H. Minsky. Selective and locally controlled transport of privileges. *ACM Transactions on Programming Languages and Systems*, 6(4):573–602, October 1984.

[24] Rajeev Motwani, Rina Panigrahy, Vijay A. Saraswat, and Suresh Ventkatasubramanian. On the decidability of accessibility problems (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 306–315. ACM Press, May 2000.

[25] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

[26] Charles P. Pfleeger. *Security in Computing*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3 edition, 2003.

[27] Ravi S. Sandhu. The schematic protection model: Its definition and analysis for acyclic attenuating systems. *Journal of the ACM*, 35(2):404–432, 1988.

[28] Ravi S. Sandhu. Expressive power of the schematic protection model. *Journal of Computer Security*, 1(1):59–98, 1992.

[29] Ravi S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Computer Society Press, May 1992.

[30] Ravi S. Sandhu. Undecidability of the safety problem for the schematic protection model with cyclic creates. *Journal of Computer and System Sciences*, 44(1):141–159, February 1992.

[31] Jon A. Solworth and Robert H. Sloan. A layered design of discretionary access controls with decidable safety properties. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 2004.

[32] Masakazu Soshi. Safety analysis of the dynamic-typed access matrix model. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 106–121. Springer, October 2000.

[33] Masakazu Soshi, Mamoru Maekawa, and Eiji Okamoto. The dynamic-typed access matrix model and decidability of the safety problem. *IEICE Transactions on Fundamentals*, E87-A(1):190–203, January 2004.