Brewer-Nash Scrutinised: Mechanised Checking of Policies featuring Write Revocation

Alfredo Capozucca
Computer Science
University of Luxembourg
Esch-sur-Alzette, Luxembourg
alfredo.capozucca@uni.lu

Maximiliano Cristiá
CIFASIS
Universidad Nacional de Rosario
Rosario, Argentina
cristia@cifasis-conicet.gov.ar

Ross Horne
Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
ross.horne@strath.ac.uk

Ricardo Katz
CIFASIS CONICET
Rosario, Argentina
katz@cifasis-conicet.gov.ar

***Abstract*—This paper revisits the Brewer-Nash security policy model inspired by ethical Chinese Wall policies. We draw attention to the fact that write access can be revoked in the Brewer-Nash model. The semantics of write access were underspecified originally, leading to multiple interpretations for which we provide a modern operational semantics. We go on to modernise the analysis of information flow in the Brewer-Nash model, by adopting a more precise definition adapted from Kessler. For our modernised reformulation, we provide full mechanised coverage for all theorems proposed by Brewer & Nash. Most theorems are established automatically using the tool {*log*} with the exception of a theorem regarding information flow, which combines a lemma in {*log*} with a theorem mechanised in Coq. Having covered all theorems originally posed by Brewer-Nash, achieving modern precision and mechanisation, we propose this work as a step towards a methodology for automated checking of more complex security policy models.**

***Index Terms*—security policies, information flow, information integrity policy, revocation, set theory, automated verification**

I. Introduction

The Brewer-Nash security policy model, inspired by Chinese Wall policies used to manage conflicts of interest particularly in the financial sector, was originally communicated at S&P'89 [1]. Chinese Walls remain as much a feature of modern businesses, as when Brewer & Nash motivated their work, with the ongoing high-profile insider-trading case of Joe Lewis highlighting the importance of being able to provide evidence that adequate policies were adhered to.[1] In computer systems, Chinese Walls have been adopted since they allow freedom of choice initially, until too much information is requested. There are established implementations of Chinese Wall policies for Unix [2], and the progression of such policies to the Xen hypervisor, where multiple organisations may share the same hardware, was almost inevitable [3].

Brewer & Nash deliberately designed their security policy model such that features are reminiscent of the Bell-LaPadula security policy model [4], that informed most lattice-based security policies. These policy models are schemes for security policies that maintain confidentiality (and sometimes integrity) of information by permitting or denying certain flows through a system. While Bell-LaPadula, which originated in policies typical of the military, would permit flows from low to high classification of objects, Brewer & Nash proposed a flat structure, more typical of companies that do not have a common administrative authority.

A distinctive feature of Chinese Wall policies, sometimes referred to as ethical policies, is that they provide some mechanism for explicitly indicating conflicts-of-interest (CoI), at some granularity such as a dataset containing information about a specific company. This remains a distinctive feature of more recent models of ethical policies, such as quantales of information [5]. The year Brewer & Nash communicated their model, Lin presented a compelling argument that CoI should be a relation, since, conflict-of-interests need not be transitive (for example, if two readers have a conflict of interest with an author then the two readers are not necessarily in conflict with each other) [6]. That idea is now accepted in the mainstream, e.g., the aforementioned Unix implementation features a matrix capturing the CoI relation, that need not be transitive.

Brewer and Nash were likely aware that CoI is naturally more general, since they state explicitly that "since we wish to compare it with the Bell-LaPadula (BLP) model we will adopt the latter's concepts of subjects, objects and security labels." Thus some of the restrictions in their model were more so to facilitate an easy comparison with concepts due in the Bell-LaPadula model. There are other dimensions in which the Brewer-Nash model has evolved over time, becoming increasingly complex—e.g., permitting more flexible policies, or distinguishing between permission to access and instances

[1] See, e.g., FT on insider trading: https://www.ft.com/insider-trading

Preprint. Accepted for publication at 37th IEEE Computer Security Foundations Symposium (CSF 2024)
https://csf2024.ieee-security.org/.

of access, etc. [7], [8]. The original model, due to Brewer-Nash, however still survives in textbooks covering information security [9].

In this work, we return our attention to 1989 and the original model of Brewer-Nash. While the model appears to be simple, there are some features that are not easy to grasp, or are easily missed, since they are handled in a rather implicit manner.

1) Firstly, the notion of information flow that they rely on is not, in our view, as well defined as in later papers on security policy models.
2) Secondly, the model has a rather novel feature for a security policy model: write access can be revoked, but in a rather implicit manner.

Point (1) above is an indicator that, while an appendix with proofs was provided, the rigour of the proofs conducted was perhaps not up to today's standards. This argument applies whether or not one agrees with our view above on information flow, thanks to the advances in automated tools and proof assistants. Since the automated tool for proving decidable theorems in set theory, called {*log*} (pronounced set log), has been used to fully and quickly automate the checking of the Bell-LaPadula policy model [10], it is natural to ask whether that automation can be lifted to other information integrity policies in general, and Brewer-Nash in particular in this work. What we will see is that Brewer-Nash is more complex to verify, and instead we go for a hybrid approach where, for the theorem concerning information flow {*log*} proves an invariant, and Coq is used to mechanise a proof confirming that the invariant is sufficient to establish the intended property. Thus, in summary, our contributions in this direction are:

- Tightening of the definitions given by Brewer & Nash that we felt necessary to establish their original theorems.
- A fully mechanised proof of the tightened theorems using {*log*} and Coq, with maximum work pushed to the automated {*log*} component.

The work can also be seen as a seed for a methodology that may be more generally applied to security policy models that maintain confidentiality and integrity with respect to information flows. Given that related policy models are deployed in real systems and their failure can have serious consequences, as discussed at the top of the introduction, it is important to certify the correctness of security policy models.

Returning to point (2) above, we clarify in a more explicit manner how write access is handled. In the motivating section, next, we elaborate on an example where we ask what happens when you can write to a dataset, and then request (read) access to another dataset. The example helps us see complications associated with this scenario, which we believe is the key novelty of the Brewer-Nash model compared to some more recent ethical policies. It is also a novelty with respect to key implementations, for instance the aforementioned Unix implementation of a Chinese Wall policy bypasses this feature of Brewer-Nash and instead proposes its own mechanism for write access. Observations such as this, lead us to believe that the Brewer-Nash model is not as simple as it first seems, and hence may be open to misuse without stronger certification.

Summary: Section II illustrates the key novelties and ambiguities in the operational semantics of the Brewer-Nash policy model, firstly in a simpler form ignoring sanitized data. Section III provides a complete operational semantics for Brewer-Nash policies, including sanitized data, and introduces the target properties expected of the Brewer-Nash policy model. Section IV explains how some properties are mechanised using the tool {*log*} by expressing appropriate invariants. Section V defines an appropriate notion of information flow and explains how information flow is mechanised by combining {*log*} and Coq. Section VI completes the mechanisation of all theorems by showing how the remaining theorem can be established in {*log*} via an explicit construction of an injective function. Section VII highlights how the methodology may be adapted to other policy models in the future.

II. Motivating scenarios: interpretations of access

We explain here a simple scenario in which write revocation occurs, while refreshing our knowledge about the Brewer-Nash policy model. In doing so, we examine the constraints on state transitions determining whether access is permitted to resources: namely, the *simple security rule* and the **-property*. This already draws our attention to how the complexity of the original Brewer-Nash model lies in the fact that, firstly, there is no distinction between read and write access in the state—there is only access—and, secondly, when one has access, implicitly one may read henceforth yet each write access is conditional on the state. This statement can already be interpreted in multiple ways that requires more precision to resolve. We also provide a more explicit formulation of the Brewer-Nash model that separately handles read and write access in the state, which we argue is more amenable to implementation. Note that we omit sanitized data from this initial discussion to keep to the point.

A. Diagrammatically

Consider the simple state transition illustrated in Fig. 1. This small example already demonstrates a surprising feature of the Brewer-Nash model, specifically that write permissions can be revoked. To follow the illustration observe that there are two conflict-of-interest classes (CoIC) CoI_1 and CoI_2 which set up boundaries between datasets (the rounded regions) indicated by the solid (red) lines. Intuitively, no subject should be able to hold[2] data originating from objects in datasets at either side of the red line. Each dataset has one object in this example o_1, o_2, and o_3. In a prior state, the subject s_1 can access no object, and has free will to request access from any dataset, which is permitted by Brewer-Nash. In this case, the subject has chosen to access o_2, resulting in the state to the left of Fig. 1, where the two-headed arrow ($\longleftrightarrow$) indicates that s_1 has read/write access to o_2. When in the state to the left of Fig. 1, it is impossible for s_1 to access o_1. This is the distinguishing

[2]Or influence when writes are concerned, but even that is already beyond what is clearly paid out by the Brewer-Nash model.

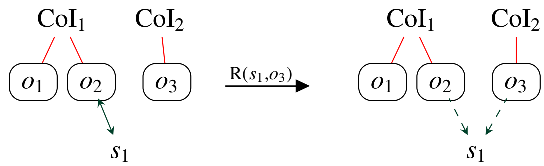

Fig. 1. A transition where write is revoked: subject s_1 requests access to o_3 (permitted by simple security), which results in write access to o_2 being revoked (due to the *-property).

feature of Chinese Wall policies in general—initially subjects have free choice to access objects, but, as accesses are granted, permissions may be restricted. In contrast, subject s_1 can request access to o_3, which lies in a separate CoIC (perhaps here, only o_1 and o_2 are owned by competitors, but o_3 is a business operating in a separate market).

Since s_1 is permitted to access o_3, the system can perform the state transition labelled with the read request in Fig. 1. However, notice that after that transition, the solid two-headed arrow ($\longleftrightarrow$) becomes a one way dotted arrow ($--\rightarrow$). These dotted arrows point from the object to the subject, indicating that, in the new state, (1) the subject has at some point been permitted access, (2) only read access is enabled in that state. Thus the subject can obtain information from the object, but not the other way round, as suggested by the direction of these arrows. The arrows therefore indicate that the write access of o_2 to s_1 has been revoked as soon as o_3 is accessed, and furthermore s_1 is never permitted write access to o_3. This write revocation is specific to a line of work more faithful in some sense to Brewer-Nash [6]–[8], and not all ethical policies.

B. Operationally

We explain more formally the machinery at play here, to understand in what sense revocation of write access is handled implicitly by Brewer & Nash, before we go on to express how write revocation may be made more explicit.

The Brewer-Nash policy model is a scheme for policies, where a specific Brewer-Nash policy is defined by security labels that are assigned to each object for the lifetime of the policy. More precisely, a Brewer-Nash policy assumes that each object o has fixed labels that assign the object a single CoIC, denoted here by coi(o), and a dataset within that CoIC, denoted ds(o). States are simply a (finite) relation between subjects and objects (the access matrix), denoted N. We can make a state transition updating N to indicate that a subject s can access an object o, only if o' is in the same dataset as o or is in a different CoIC from o. This condition on access is called the *simple security* rule.

Definition 1 (simple security)**.** *A subject s, object o and matrix N satisfy the simple security rule whenever:*

$$\forall o' : (s, o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{coi}(o) \neq \mathrm{coi}(o'))$$

Thus read access, conditional on the simple security rule, can be expressed, using the conventions of labelled transition systems, as follows.

$$\frac{\forall o' : (s, o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{coi}(o) \neq \mathrm{coi}(o'))}{N \xrightarrow{\mathrm{R}(s,o)} N \cup \{(s, o)\}} \quad \text{(iR)}$$

Stated otherwise, the above rule expresses that, s can access o if s has not accessed any other object that is in another dataset in the same CoIC as o. It is not stated clearly or expressed formally in the original paper [1], but we can assume that "access" here refers to read access specifically. The need for that clarification becomes important, given that the next property refers to read and write access.

The *-property is described informally by Brewer & Nash in terms of the capability to "read" and "write", rather than the neutral "access" of the simple security rule. More precisely, it states that write access is permitted if (1) the simple security rule holds, and (2) no subject "can read" an object in a dataset different from the one requested. The details of the operational rule for write access is open to interpretation. We see arguments for and against an interpretation where writing (disseminating and appending data) is possible before requesting read access using Eq. (iR).

One interpretation, coming from the model provided by Brewer-Nash in their appendix suggests that any access entails read access, since the formal way "s can read o" is modelled is by checking whether $(s, o) \in N'$, where N' is the state after the transition (see Axiom 6 in the original appendix of Brewer & Nash [1]). The *-property implies the simple security rule (hence checking the simple security rule is redundant as noted first by Lin [6]). This leads us to the following labelled transition for write access.

$$\frac{\forall o' : (s, o') \in N \implies \mathrm{ds}(o') = \mathrm{ds}(o)}{N \xrightarrow{\mathrm{W}(s,o)} N \cup \{(s, o)\}} \quad \text{(iRW)}$$

For example, assuming the policy given by the labels in Fig. 1, transitions $\{(s_1, o_2)\} \xrightarrow{\mathrm{W}(s_1,o_2)} \{(s_1, o_2)\}$ and $\{(s_1, o_2)\} \xrightarrow{\mathrm{R}(s_1,o_2)} \{(s_1, o_2)\}$ can be applied indefinitely at first, until the transition $\{(s_1, o_2)\} \xrightarrow{\mathrm{R}(s_1,o_3)} \{(s_1, o_2), (s_1, o_3)\}$ occurs. After that point, transitions $\{(s_1, o_2), (s_1, o_3)\} \xrightarrow{\mathrm{R}(s_1,o_2)} \{(s_1, o_2), (s_1, o_3)\}$ and $\{(s_1, o_2), (s_1, o_3)\} \xrightarrow{\mathrm{R}(s_1,o_3)} \{(s_1, o_2), (s_1, o_3)\}$ are enabled indefinitely. Yet, no write transition involving s_1 is enabled. Thus the write access of s_1 to o_2 is implicitly revoked.

Other authors have produced alternative interpretations for how write access is defined. For example, in an influential lattice-based formulation of a Chinese Wall policy model [11], it is clear that Sandhu permits write access anywhere if read access is granted nowhere. In short, according to Sandhu, subjects are also labelled, and those subjects with no read access are labelled with the bottom element in a lattice of security labels, which is below all dataset labels that are assigned to objects (which, as normal, confine confidential information in objects to their dataset). Since the *-property is generalised by Sandhu such that write is permitted upwards in a particular lattice, clearly subjects that have not yet read anything can write anywhere.

We, the authors, even are split on how to interpret the definitions of Brewer & Nash as operational rules (see their Def. 1 and Axiom 6), and given there is a split in the literature, we explore both interpretations. If we argue that granting write access does not automatically grant read access, we can employ the following rule.[3]

$$\frac{\forall o' : (s, o') \in N \implies \mathrm{ds}(o') = \mathrm{ds}(o)}{N \xrightarrow{\mathrm{W}(s,o)} N} \tag{iW}$$

This models a more permissive policy allowing write access without granting read access, as indicated by not updating N. Thus, one can write freely to all datasets, until one dataset is read from; at which point, write is (implicitly) revoked to all other datasets (by the *-property). Transitions illustrating this sequence of operations are presented in Fig. 2, where the head of the arrow depicting write-only access points in the opposite direction from read-only access seen previously.

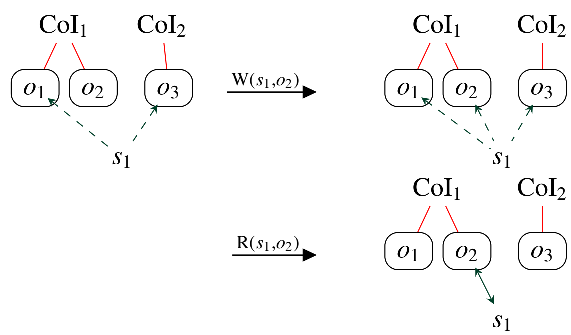

Fig. 2. Transitions in a permissive interpretation of Brewer-Nash policies where write-only access can be requested anywhere initially. Notice that write access is revoked when read access is requested.

Eq. (iW) allows each subject to have a phase where they use their own knowledge (to generate reports based on public information, for example) and push it to any dataset. This rule appears to be permissible from the perspective of information confidentiality, in the sense that we expect that secrets held within objects (e.g., confidential information about clients that a trader must respect in the financial sector) will not be leaked to unintended objects by writing freely before reading anything confidential. When a subject writes without reading, the subject can only write information known already before entering the ecosystem—only the subject's knowledge or special data processing skills are given away. There is therefore no violation of a nondisclosure agreement with a company associated with a dataset, since only the subject's own information is written, which is not subject to confidentiality constraints. This keeps Brewer-Nash in line with the confidentiality aims of Bell-LaPadula avoiding inadvertent declassification, but cannot prevent entirely people going rogue and, for example, distributing information outside the system (c.f. Teixeira's Pentagon leaks via screenshots posted on Discord).

This write-only (aka. append) phase precedes the phase of read/write editing within one dataset only. And, finally, there is the read-only phase already discussed, where the information system assists the subject in ensuring they do not violate conflicts-of-interests in their ultimate decision making. Confidentiality is therefore preserved (formalised in Sec. V).

Both interpretations of write access discussed above have their merits. Furthermore, we will find that it does not create problems going forward to have both rules coexisting in one model. Therefore, either rule, may be selected when implementing a system enforcing a policy. Indeed, some objects may be governed by Eq. (iRW) and other by Eq. (iW) within the same system without compromising security.

C. Explicitly

From the above model we can see that the Brewer-Nash model is assuming that write is an atomic action, in the sense that each time a subject would like to write anything the *-property must be checked before the write access is granted, and furthermore we must be sure that the write access completes before another operation is applied (or at least before a read request to another dataset is granted, in a system with more advanced awareness of concurrency). This approach, by Brewer & Nash to write operations comes at a cost (in terms of concurrency control and logical checks), due to the need to check the *-property repeatedly while locking certain other operations, rather than referring to an access control matrix. That complexity is perhaps among the reasons why policies such as Unix Chinese Wall policies [2] do not implement write access using the *-property at all.

This observation leads us naturally to a more explicit approach that we introduce in this work, which is to include an additional write access matrix that makes any previously granted write access explicit until the *-property is violated. Read and write access are formalised via the operational rules, explained next, that refer to a write access matrix W.

As explained when discussing Eq. (iW), an interpretation of Brewer-Nash is that write access may be write-only, and hence N is not updated as in the following rule.

$$\frac{\forall o' : (s, o') \in N \implies \mathrm{ds}(o') = \mathrm{ds}(o)}{N, W \xrightarrow{\mathrm{W}(s,o)} N, W \cup \{(s, o)\}} \tag{xW}$$

Recall that, alternatively, a policy may insist that read access is granted whenever write access is granted (c.f., Eq. (iRW)), which is a legitimate interpretation of the partial definitions of Brewer & Nash.[4] In this case, observe below that both the read

[3]The rule can be formulated without the lattice machinery of Sandhu, who anyway does not formalise state transitions. Sandhu models snapshots as a lattice, and suggests informally that privileges may float up depending on what operational behaviour is desired. The objective of Sandhu is to explain that Chinese Wall policy models can be cast in the same light as the Bell-Lapadula policy models when it comes to the simple security property and *-property; that is, subjects can read below and write above in a suitable lattice structure, depending on labels assigned to subjects as well as objects.

[4]Derived from the explanations of Brewer & Nash that: (1) $(s, o) \in N$ means "s has, or has had, access to object o." in a passage that can be interpreted as generically describing all types of access rather than read access specifically, and; (2) there is no formal mention of read, except "has read" in the *-property and hence access is the only candidate; (3) the state N' after a write operation should contain the subject and object to which write access is granted.

access and write matrix are updated, where in what follows we define $W' = W \setminus \{(s, o') \colon \text{ds}(o') \neq \text{ds}(o)\}$.

$$\frac{\forall o' \colon (s, o') \in N \implies \text{ds}(o') = \text{ds}(o)}{N, W \xrightarrow{\text{W}(s,o)} N \cup \{(s, o)\}, W' \cup \{(s, o)\}} \quad \text{(xRW)}$$

The updated write access matrix W' explicitly revokes write access, to any object in another dataset when the above rule is applied. This caters for the possibility that write access may have been granted to another dataset, which is clearly possible if Eq. (xW) is allowed to coexist with Eq. (xRW) above.[5]

Since, once granted, read access is recorded in N and write access is recorded in W, we need not check the *-property each time a read or write occurs, as in the original Brewer-Nash model. Instead, we simply consult the access matrices N or W respectively and permit the operation if there is an appropriate entry. This observation leads us to the following cheap rule for access, while other rules need only be appealed to if the rules below fail to grant access.

$$\frac{(s, o) \in N}{N, W \xrightarrow{\text{R}(s,o)} N, W} \qquad \frac{(s, o) \in W}{N, W \xrightarrow{\text{W}(s,o)} N, W} \quad \text{(access matrix)}$$

The interesting question is what happens when read access is requested in another dataset in a different CoIC from where write has been granted, as per Fig. 1. Neither of the cheap access matrix lookup rules apply. In this more explicit model, in order to avoid a violation of the *-property, it is important also to check that the new read does result in the *-property being violated for some write that has already been granted. If it does then either we:

- deny the read operation, or
- explicitly revoke all offending write accesses in W.

In terms of user experience, indeed it seems appropriate to ask the user (or run some conflict resolution algorithm), since it may be that the subject welcomes the warning and decides that they prefer not to read the object in the new dataset, and instead retain read-write access to their current dataset.

The two options above, correspond to the following operational rule. In the following, $W \setminus \{(s, o') \colon \text{ds}(o') \neq \text{ds}(o)\}$ explicitly revokes any offending write accesses.

$$\frac{\forall o' \colon (s, o') \in N \implies (\text{ds}(o') = \text{ds}(o) \vee \text{coi}(o') \neq \text{coi}(o))}{N, W \xrightarrow{\text{R}(s,o)} N \cup \{(s, o)\}, W \setminus \{(s, o') \colon \text{ds}(o') \neq \text{ds}(o)\}} \quad \text{(xR)}$$

Thus the rule above can be used to more explicitly realise the transition in Fig. 1, as follows.

$$\{(s_1, o_2)\}, \{(s_1, o_2)\} \xrightarrow{\text{R}(s_1,o_3)} \{(s_1, o_2), (s_1, o_3)\}, \emptyset$$

Similarly, the operations in Fig. 2 consist of a write operation followed by a read that revokes write access to two datasets.

$$\begin{aligned} \emptyset, \{(s_1, o_1), (s_1, o_3)\} &\xrightarrow{\text{W}(s_1,o_2)} \emptyset, \{(s_1, o_1), (s_1, o_2), (s_1, o_3)\} \\ &\xrightarrow{\text{R}(s_1,o_2)} \{(s_1, o_2)\}, \{(s_1, o_2)\} \end{aligned}$$

[5]This will also be possible even if Eq. (xW) were forbidden, once we introduce sanitized data in the next section. N.B. "x" abbreviates "explicit".

An alternative is to allow a read access, without revoking write access, under conditions ensuring that the *-property will be preserved for everything already recorded in W. The condition is that all objects that the subject can write to according to W must be in the same dataset as where read access is requested. This condition concerning W is of course in addition to the standard assumption that the simple security rule holds with respects to objects the subject can read from. This restrictive read rule is expressed as follows.

$$\frac{\begin{array}{c}\forall o' \colon (s, o') \in N \implies \text{ds}(o') = \text{ds}(o) \vee \text{coi}(o') \neq \text{coi}(o) \\ \wedge\ (s, o') \in W \implies \text{ds}(o) = \text{ds}(o')\end{array}}{N, W \xrightarrow{\text{R}(s,o)} N \cup \{(s, o)\}, W} \quad \text{(xR*)}$$

The "*" in the rule name above highlights the additional check required to preserve the *-property. We will return to the rules above in Eq's (xR) and (xR*) in detail in subsequent sections, since they are novel rules, and it is not immediately obvious that they do in fact preserve the *-property. Thus our explicit rules benefit from the ensuing verification.

Notice that if Eq. (xR*) applies, then Eq. (xR) also applies and has the same effect. However, if a policy features both rules then, whenever Eq. (xR*) is not enabled it is possible to trigger a suitable warning that explains to the subject that reading the object in question is going to result in write access being revoked somewhere else. Thus, distinguishing these transitions helps to demarcate an important transition in the life of a subject. Eq. (xR*) ensures a subject can continue to read and write within a dataset; while if only Eq. (xR) applies then a subject induces a state transition that may prevent the subject from writing again.

III. Full definitions and theorems to cover

Here we collate the key definitions and theorems that we mechanise in this work. In subsequent sections, we explain the theorems in more detail including how $\{log\}$ and Coq are used to mechanise them. The theorems are the four theorems stated by Brewer-Nash in the order that they appear in that work to facilitate a close comparison. These reformulated theorems make use of the modernised notation and definitions from the previous section.

A. The full explicit model with sanitized data

For complete coverage of Brewer-Nash we introduce the concept of sanitized data, that didn't play a role in the previous section. The explicit rules from the previous section are expanded and collated in Fig. 3.

Sanitized data refers to public knowledge or, perhaps, data checked and approved to be distributed within the system regulated by the policy. A special dataset, denoted by *Yo*, contains sanitized objects, which is the only dataset in a conflict of interest class which we call Sanitized (see Fig. 4). The *-property in the presence of sanitized data is formulated as follows.

$$\frac{\forall o' : (s,o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{ds}(o') = Yo)}{N, W \xrightarrow{\mathrm{W}(s,o)} N, W \cup \{(s,o)\}}\ xW \qquad \frac{(s,o) \in W}{N, W \xrightarrow{\mathrm{W}(s,o)} N, W}\ mW$$

$$\frac{\mathrm{ds}(o) \neq Yo \wedge \forall o' : (s,o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{ds}(o') = Yo)}{N, W \xrightarrow{\mathrm{W}(s,o)} N \cup \{(s,o)\}, (W \setminus \{(s,o') : \mathrm{ds}(o') \neq \mathrm{ds}(o)\}) \cup \{(s,o)\}}\ xRW \qquad \frac{\mathrm{ds}(o) = Yo \wedge \forall o' : (s,o') \in N \implies \mathrm{ds}(o') = Yo}{N, W \xrightarrow{\mathrm{W}(s,o)} N \cup \{(s,o)\}, W \cup \{(s,o)\}}\ xRW\bot$$

$$\frac{\mathrm{ds}(o) \neq Yo \wedge \forall o' : (s,o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{coi}(o') \neq \mathrm{coi}(o))}{N, W \xrightarrow{\mathrm{R}(s,o)} N \cup \{(s,o)\}, W \setminus \{(s,o') : \mathrm{ds}(o') \neq \mathrm{ds}(o)\}}\ xR \qquad \frac{\mathrm{ds}(o) = Yo}{N, W \xrightarrow{\mathrm{R}(s,o)} N \cup \{(s,o)\}, W}\ xR\bot$$

$$\frac{(\forall o' : \ ((s,o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{coi}(o') \neq \mathrm{coi}(o))) \wedge ((s,o') \in W \implies \mathrm{ds}(o') = \mathrm{ds}(o)))}{N, W \xrightarrow{\mathrm{R}(s,o)} N \cup \{(s,o)\}, W}\ xR* \qquad \frac{(s,o) \in N}{N, W \xrightarrow{\mathrm{R}(s,o)} N, W}\ mR$$

Fig. 3. Explicit rules for Brewer-Nash policies, including sanitized data.

Definition 2 (*-property)**.** *A subject s, object o and matrix N satisfy the *-property whenever:*

$$\forall o' : (s,o') \in N \implies (\mathrm{ds}(o) = \mathrm{ds}(o') \vee \mathrm{ds}(o') = Yo)$$

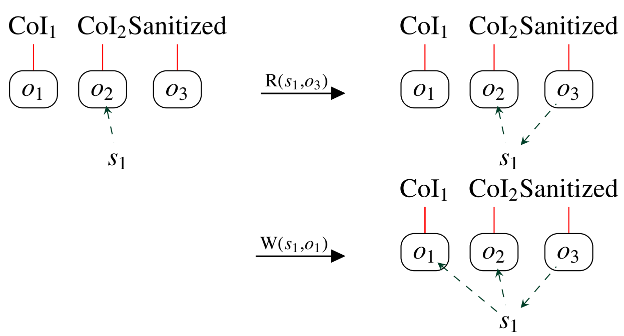

Fig. 4. Even after writing to a private dataset, the user can read from sanitized data. Furthermore, reading sanitized data does not prevent writing to further datasets.

Consider the example in Fig. 4 to understand how rules behave differently when sanitized data is involved. The read transition in that figure uses rule $xR\bot$, which allows objects in the sanitized dataset to be read without revoking write access anywhere. In contrast, rule xR does not apply to this example, and if the condition $\mathrm{ds}(o) \neq Yo$ were dropped from xR, then xR would revoke write access where it is not necessary to do so. Fig. 4 also illustrates that write access can still be freely requested elsewhere after reading. This behaviour contrasts to read access to data that is not sanitized, which always blocks write operations in other datasets from that point onward.

The notation $\bot$ in rule name signals the consistency of Brewer-Nash with the lattice-based interpretation of Sandhu [11], mentioned in the previous section. Sandhu assigns for sanitized objects, and also subjects who have not read from a dataset that is not sanitized, the bottom security label in a lattice of labels. Therefore, since in Sandhu's lattice-based model reading is permitted downwards and writing upwards, subjects with the bottom security label can still request read access to sanitized data while writing anywhere.

Consider now the example in Fig. 5. This shows an exceptional revocation behaviour associated with sanitized data. Initially, subject s_1 can read and write to a sanitized object. The subject can, by calling rule xRW, induce a state change where their write access to the sanitized object is revoked. Observe that, in contrast to Fig. 1 which did not involve sanitized data, write access is still possible for object o_2.

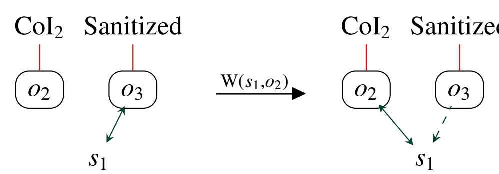

Fig. 5. Only sanitized data has the property that if read-write access has been granted, then read-write access may still be granted in another dataset.

B. The four Brewer-Nash Theorems

We state the four theorems proposed by Brewer & Nash in their original form and using more modern precision. All of these theorems are stated with respect to the labelled transition system of the explicit model generated by the rules in Fig. 3. Thus, all theorems in this section range over any policy consisting of a fixed set of subjects, objects, CoICs, mappings $\mathrm{ds}(\,.\,)$, and $\mathrm{coi}(\,.\,)$, and satisfy axioms:

$$\forall o_1, o_2 : \mathrm{ds}(o_1) = \mathrm{ds}(o_2) \implies \mathrm{coi}(o_1) = \mathrm{coi}(o_2) \qquad (1)$$

$$\forall o : \mathrm{ds}(o) = Yo \Leftrightarrow \mathrm{coi}(o) = \text{Sanitized} \qquad (2)$$

Axiom (2), above, ensures that, even when sanitized data is considered, whenever a subject, object and matrix satisfy the *-property then they satisfy the simple security rule. This helps explain why the simple security rule is redundant in the rules in Fig. 3 involving write access.

For the first theorem, Brewer & Nash state: "Once a subject has accessed an object the only other objects accessible by that subject lie within the same company dataset or within a different conflict-of-interest class."

This essentially says that, w.r.t. Fig. 3, it is an invariant that all subject-object pairs in N satisfy the simple security rule. A property I is proven to be an invariant by establishing that if $I(N, W)$ holds and, for any action α and state N', W', if $N, W \xrightarrow{\alpha} N', W'$, then $I(N', W')$ holds. Furthermore, the initial state $\emptyset, \emptyset$ must satisfy I. This ensures that the property holds in all states reachable from the initial state.

Theorem 1. *The following is an invariant for states N, W: for all $(s, o) \in N$, s, o and N satisfy the simple security rule.*

The second theorem of Brewer & Nash states: "A subject can at most have access to one company dataset in each conflict-of-interest class." This is captured formally by the consequent of Theorem 2 below.

Theorem 2. *Consider any state N, W satisfying the property that, for all $(s, o) \in N$, s, o and N satisfy the simple security rule. For any subject s and objects o_1 and o_2, if $(s, o_1) \in N$ and $(s, o_2) \in N$ and* $\mathrm{coi}(o_1) = \mathrm{coi}(o_2)$*, then* $\mathrm{ds}(o_1) = \mathrm{ds}(o_2)$.

The premise of the Theorem 2, was absent in the formulation due to Brewer & Nash. The premise clarifies that the Theorem holds for any state satisfying the invariant established in Theorem 1. Hence the consequent of Theorem 2 is itself preserved by all transitions, and hence is itself an invariant.

The third theorem of Brewer & Nash states: "If for some conflict-of-interest class X there are X_v company datasets then the minimum number of subjects which will allow every object to be accessed by at least one subject is X_v."

In the above, "accessed by" is interpreted as read access as recorded by N. This leads to the following formalisation of this property as an invariant, where $|A|$ denotes the cardinality of set A.

Theorem 3. *Let $\mathcal{S}$ be some set of subjects and $\mathcal{D}$ be some set of datasets fixed for the policy. Also, for any CoIC X, let $X_v = \{Y \in \mathcal{D} : \exists o.\mathrm{ds}(o) = Y \wedge \mathrm{coi}(o) = X\}$.*

The following is an invariant for states N, W. For any CoIC X, if for all datasets $Y \in X_v$ there exists subject s and object o such that $(s, o) \in N$ and $\mathrm{ds}(o) = Y$, then $|X_v| \leq |\mathcal{S}|$.

Theorem 4 of Brewer & Nash is formulated as: "The flow of unsanitized information is confined to its own company dataset; sanitized information may however flow freely throughout the system."

The notion of flow is not defined by Brewer & Nash, and we defer the definition that we will employ until Sec. V. However, we assume here that there is some well-defined notion of flow of information from one object o_1 to another object o_2, starting from a given state N, W, denoted $o_1 \rightsquigarrow o_2$ in the following theorem.

Theorem 4. *The following is an invariant for states N, W: For objects o and o', if $o \rightsquigarrow o'$ starting in N, W, then $\mathrm{ds}(o) = Yo$ or $\mathrm{ds}(o) = \mathrm{ds}(o')$.*

Brewer & Nash, in their proof of Theorem 4, we believe, just jump to their desired conclusion by defining a relation that satisfies a given property. Theorem 4 turns out to be the trickiest theorem to define more precisely and prove, as we explain in detail in Sec. V.

IV. Automated reasoning about invariants in {*log*}

In this section we show how we use {*log*} to formally specify and verify that the simple security rule and *-property are invariants of our formal interpretations of Brewer-Nash policies. This section covers the mechanisation of Theorem 1 and Theorem 2 and lays essential groundwork towards the mechanisation of Theorem 4 (Sec. V).

The tool {*log*} is a constraint logic programming (CLP) language and satisfiability solver implemented in Prolog where finite sets are first-class citizens [12], [13]. The tool implements decision procedures for several fragments of set theory and set relation algebra [14]–[18]. A few in-depth empirical studies provide evidence that {*log*} is able to solve non-trivial problems, e.g. [19], [20]. On top of its CLP language, {*log*} provides a state machine specification language (SMSL) inspired in the B notation [21]. A verification condition generator (VGC) can then be used to automatically generate verification conditions (VC) ensuring that the state machine verifies some properties [22, Sect. 11]. {*log*} inherits many Prolog features. For instance, variables must begin with a capital letter; the main program building block are predicates expressed as Horn clauses of the form $\boxed{\mathit{head}(\mathit{params}) \text{ :- } \mathit{body}.}$ where *body* is a {*log*} formula (note the dot at the end of *body*).

We describe here the {*log*} formalisation of the Brewer-Nash policy model, available in the companion replication package [23]. The set of objects of the system (*Objects*), the function mapping objects onto security classes (L) and the dataset containing sanitized information (*Yo*) and its conflict of interest class (*Xo*, aka. Sanitized in the previous section) are introduced as parameters of the model.

$$\mathsf{parameters}([\mathit{Objects}, L, \mathit{Yo}, \mathit{Xo}]).$$

The state space of the system is given by two state variables: N, denoting the current read accesses for each subject; and W, denoting the current write accesses for each subject.

$$\mathsf{variables}([N, W]).$$

Axioms are used to state properties of parameters. For example, L is a function whose domain is *Objects*.[6]

$$\mathsf{axiom}(\mathit{axiomL}).$$
$$\mathit{axiomL}(L, \mathit{Objects}) \text{ :- } \mathsf{pfun}(L) \wedge \mathsf{dom}(L, \mathit{Objects}).$$

Above, pfun is a {*log*} constraint stating that its argument is a function whereas dom states that *Objects* is the domain of L. Since finite sets are the main data structure in {*log*}, {*log*} admits sets of ordered pairs, i.e., binary relations.

Relations N and W, as we will shortly see, are augmented with the labels associated with objects (the dataset and CoIC) because {*log*} proofs become faster. This is a difference between purely theoretical considerations such as those discussed in Sec. II and III and the representation of those concepts in an automated tool. Invariants are used to ensure that labels in N and W are subject to the conditions imposed on L.

[6] Instead of using the exact {*log*} ASCII notation, we rather use a more math-oriented one thus avoiding some syntactic nuisances.

State invariants are given as predicates that depend on parameters and state variables. An invariant property appealing to the simple security rule is encoded as follows in {*log*}.

$$\begin{aligned}
&\mathsf{invariant}(\mathit{simpSec}).\\
&\mathit{simpSec}(N) \text{ :-}\\
&\quad \forall (S_1,(O_1,(C_1,D_1))),(S_2,(O_2,(C_2,D_2))) \in N :\\
&\qquad S_1 = S_2 \implies (C_1 \neq C_2 \lor D_1 = D_2).
\end{aligned}$$

That is, N is a set of ordered pairs of the form $(S,(O,\ell))$ where S is a subject, O an object and ℓ a security label (which in turn is of the form (C,D) for some CoIC C and dataset D). Then, if $(S,(O,\ell)) \in N$ it means that subject S is accessing object O in read mode and the security label of O is ℓ. In this way, *simpSec* states that, if a subject is accessing two or more objects in read mode, their CoIC are different or their datasets are the same. It is easy to check that *simpSec* is a faithful formalisation of the invariant in Theorem 1.

Note that in *simpSec* the quantification is a restricted quantification made with ordered pairs instead of variables. A restricted quantification is a formula of the form $\forall x \in A : \phi$ equivalent to $\forall x (x \in A \implies \phi)$. The presence of ordered pairs as quantified expressions is a distinctive feature of {*log*} which allows us to increase the decidable fragment of formulas featuring restricted quantifiers [18].

An invariant preserving the *-property for all pairs in W is defined as follows. The preservation of this invariant will be used in Sec. V as part of the proof of Theorem 4.

$$\begin{aligned}
&\mathsf{invariant}(\mathit{starProp}).\\
&\mathit{starProp}(\mathit{Yo},N,W) \text{ :-}\\
&\quad \forall (S_1,(O_1,(C_1,D_1))) \in N; (S_2,(O_2,(C_2,D_2))) \in W :\\
&\qquad S_1 = S_2 \implies (D_1 = D_2) \lor D_1 = \mathit{Yo}.
\end{aligned}$$

That is, W has a similar structure to N although its interpretation is that subject s is accessing object o in write mode. In this way, *starProp* states that if a subject accesses some objects in read mode and others in write mode then they must belong to the same dataset or the subject is reading only sanitized information (*Yo*). As with N, the property where $(S,(O,\ell)) \in W$ implies $(O,\ell) \in L$, is stated as an invariant.

After giving all the invariants the initial state can be defined. This states that no access is granted to any subject initially.

$$\begin{aligned}
&\mathsf{initial}(\mathit{init}).\\
&\mathit{init}(N,W) \text{ :- } N = \emptyset \land W = \emptyset.
\end{aligned}$$

Now state transitions, called operations, are specified. Operations are predicates depending on at least one state variable. If state variable X is changed during the transition its new value is denoted by X'. Operations are given by specifying their pre- and post-conditions as {*log*} formulas. The first operation we show corresponds to a model where read access is granted only if simple security and *-property are preserved. This is called **-property read*, denoted here *spRead*, and corresponding to *xR** in Fig. 3 (it also incorporates *xR*⊥).

$$\begin{aligned}
&\mathsf{operation}(\mathit{spRead}).\\
&\mathit{spRead}(\mathit{Xo},\mathit{Yo},L,N,W,S,O,N') \text{ :-}\\
&\quad (S,(O,(C,D))) \notin N\\
&\quad \land \mathsf{ApplyTo}(L,O,(C,D))\\
&\quad \land \forall (S_1,(O_1,(C_1,D_1))) \in N : && (\mathit{pre}_{ss})\\
&\qquad S_1 = S \implies (C_1 \neq C \lor D_1 = D)\\
&\quad \land (D = \mathit{Yo} && (\mathit{pre}_{sp})\\
&\qquad \lor \forall (S_1,(O_1,(C_1,D_1))) \in W :\\
&\qquad\quad S_1 = S \implies D_1 = D)\\
&\quad \land N' = \{(S,(O,(C,D)))\ /\ N\}. && (\mathit{post})
\end{aligned}$$

In the above, *spRead* takes *Xo*, *Yo*, the state variables, a subject (S) and an object (O), and returns N', i.e. the new value of N. All but the last line are pre-conditions. The first precondition ensures that S has not opened O for reading. The second precondition states that the security label of O is (C,D) by using the {*log*} constraint ApplyTo. Pre-condition pre_{ss} checks the simple security rule (Def. 1). Pre-condition pre_{sp} is necessary to preserve the *-property. It ensures that O is a sanitized object or that the dataset of the objects that S has write access to coincides with the dataset of O. If all these hold, then $(S,(O,(C,D)))$ is added to N by means of an extensional set constructor available in {*log*}. In effect, since $\{X\ /\ A\}$ is interpreted as $\{X\} \cup A$, then N' is equal to N plus the ordered pair in question. Given that *spRead* grants read permission, W' is not included as an argument, thus $W = W'$.

The {*log*} code includes two more variants of the read operation. In one of them, called *wkRead*, the pre_{sp} precondition is not present. Operation *wkRead* can result in a conflict of interests, as illustrated in Figure 6. Notice that the accesses of s_1 and s_2 are initially with respect to the simple security invariant, since o_1 and o_3 are in different CoI classes. Notice that the initial accesses of s_1 and s_2 respect the simple security invariant, since o_1 and o_3 are in different CoI classes. After s_1 requests this excessively "weak" read access to o_3 (since only the simple security precondition holds) s_1 can access privileged information stored in o_1 which should not be allowed as s_1 already gained access to o_2. The access to privileged information is facilitated by s_2 as it may transfer information from o_1 and store it into o_3. Such shortcomings are identified by {*log*} when reporting that *wkRead* does not preserve *starProp* when attempting its mechanised verification.[7]

A variant of read that does preserve our invariants, called *revoke read*, named *rvkRead* in the {*log*} code and *xR* in Fig. 3, does not contain pre_{sp}, as *wkRead*. Instead, however, it updates W by revoking all the write accesses of S to objects whose

[7]Guidelines on how to reproduce the automated verification are provided in the replication package [23].

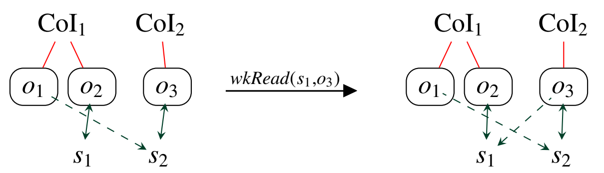

Fig. 6. A conflict of interests resulting from executing the *wkRead* operation.

dataset is different from the dataset of O. The update on W is specified as follows.

$$(D = Yo \land W' = W$$
$$\lor D \neq Yo$$
$$\land \mathsf{diff}(W, \{(S_1,(O_1,(C_1,D_1))) \in W \mid S_1 = S \land D_1 \neq D\}, W'))$$

Above, $\mathsf{diff}(A,B,C)$ is a $\{log\}$ constraint interpreted as $C = A \setminus B$. W' must be an argument of *rvkRead*. The condition $D \neq Yo$ ensures that reading sanitized data does not result in write access being revoked.

The specification of the operation xW in Fig. 3, called *write* in $\{log\}$, is as follows.

$$\begin{aligned}
&\mathsf{operation}(write).\\
&write(Yo,L,N,W,S,O,W') \text{ :-}\\
&\quad (S,(O,(C,D))) \notin W\\
&\quad \land \mathsf{ApplyTo}(L,O,(C,D))\\
&\quad \land \forall (S_1,(O_1,(C_1,D_1))) \in N : \qquad (pre^w_{sp})\\
&\qquad S_1 = S \implies (D_1 = D \lor D_1 = Yo)\\
&\quad \land W' = \{(S,(O,(C,D))) \mathbin{/} W\}.
\end{aligned}$$

In order to ensure that *-property is preserved after *write*, pre^w_{sp} checks that the objects already opened in read mode by S belong to the dataset of O or all of them contain only sanitized information.

There exists a second variant of the write operation, called *read-write*, written *readWrite*, covering both xRW and $xRW\bot$ in Fig. 3. While this operation ensures that *-property is preserved similarly as *write*, it also updates N and W. The new request—i.e. $(S,(O,(C,D)))$— is added to N while W is updated either by revoking all the write accesses of S to objects whose dataset is different from the dataset of O—as done in operation *rvkRead* shown earlier—or by adding the request when S is accessing only sanitized information.

Once the operations have been given, the VCG is run thus generating a new file containing a number of VCs. Among the most important VCs are the so-called *invariance lemmas*. An invariance lemma is a VC of the form $I \land T \implies I'$, where I is an invariant, T an operation and I' is $I[\forall v \in st(I) : v \mapsto v']$ with $st(I)$ the set of state variables of I. Informally, an invariance lemma states that if an invariant holds in some state and an operation is executed, the invariant holds in the next state. In other words, the invariant is *preserved* by the operation. Given that $\{log\}$ is a satisfiability solver, invariance lemmas generated by the VCG take a negated form: $\neg(I \land T \implies I')$. Hence, if $\{log\}$ determines that the above is unsatisfiable, the inner formula is a theorem. The VCG generates an invariance lemma for each invariant and operation. For example:

$$\begin{aligned}
&\neg(starProp(Yo,N,W) \land write(Yo,L,N,W,S,O,W')\\
&\quad \implies starProp(Yo,N,W'))
\end{aligned}$$

Note that in the consequent N appears rather than N', since N remains unchanged in *write* (N' is not one of its arguments).

Since an invariance lemma trivially holds if either the invariant or the operation are unsatisfiable, $\{log\}$ also generates VCs ensuring that the initial state satisfies every invariant and that all operations are satisfiable. The VCG generates also a predicate calling all the VCs. Then, when the user runs this predicate $\{log\}$ attempts to discharge all the VCs.

Besides the standard VCs concerning the verification of state machines, $\{log\}$ users can define their own VCs in the form of clauses declared as theorem. As with invariance lemmas, $\{log\}$ theorems have to be written in negated form. Each such declaration is included by the VCG as a VC. User-defined theorems have been used to prove, for instance, Theorems 1 and 2. For Theorem 2 we first define a clause with its consequent ($t2$) and then we declare a theorem ($theorem2$) where *simpSec* is the hypothesis required to prove $t2$. We can use *simpSec* as an hypothesis because we have proved that it is a state invariant. The fact that *simpSec* is enough to prove these theorems shows the importance of finding the right invariants for a model and, more specifically, the importance of *simpSec* and *starProp* in this context. This is further stressed in Sec. V.

V. Information Flow

A goal of the Brewer-Nash policy model is to ensure that sensitive information flows within its intended context. This section explains the mechanisation of Theorem 4 that establishes confidentiality with respect to certain information flows resulting from read and write operations. We formalise information flow, and also prove that knowing that the *-property is an invariant and read access monotonically increases is sufficient to preserve the confidentiality property expressed by Brewer & Nash in Theorem 4. We settle for a definition of information flow based on the earliest definition of information flow in the context of Chinese Wall policies that is precise enough for our purposes [7].

A. Defining information flows

When defining information flow we make use of big-step labelled transitions, that perform zero or more transitions before the given label occurs. This permits some operations that are not part of the given flow to occur in between operations that are part of the flow.

Definition 3 (big-step transition). $N_0, W_0 \overset{\alpha_n}{\Longrightarrow} N_{n+1}, W_{n+1}$ *whenever there exists* $N_0, W_0, N_1, W_1, \ldots N_{n+1}, W_{n+1}$ *and for all* $i \in [0 \ldots n]$, $N_i, W_i \xrightarrow{\alpha_i} N_{i+1}, W_{i+1}$ *(according to Fig. 3).*

We can now express formally a suitable notion of information flow, inspired by Kessler [7].

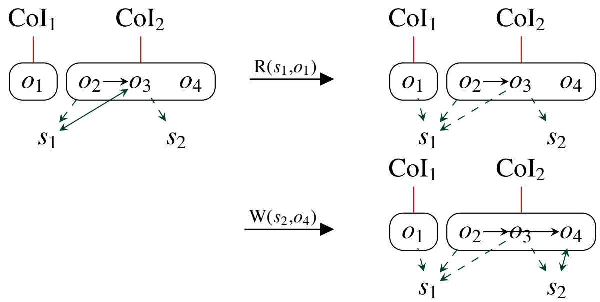

Fig. 7. Part of a flow between o_2 and o_4. Prior steps can establish $o_2 \rightsquigarrow o_3$, via s_1. Action $\mathrm{W}(s_2, o_4)$ completes the flow $o_2 \rightsquigarrow o_4$, despite write access to o_3 being revoked, since o_3 may already be influenced by o_2.

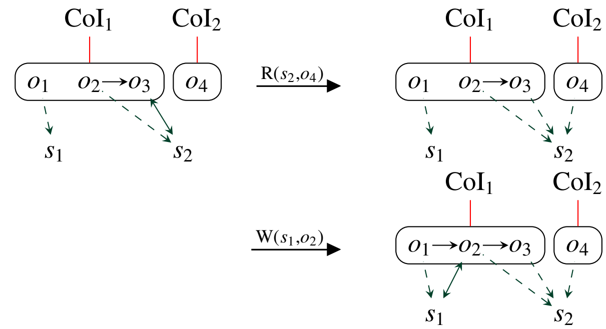

Fig. 8. These transitions do not result in a flow from o_1 to o_3.

Definition 4 (information flow)**.** *Starting in state N_1, W_1, information can flow from object o_1 to object o_{n+1}, written $o_1 \rightsquigarrow o_{n+1}$ whenever:*

$$\exists s_1, \ldots, s_n; o_2, \ldots, o_n; N_2, \ldots, N_{2n+1}; W_2, \ldots, W_{2n+1} :$$
$$\forall i : 1 \leq i \leq n \implies$$
$$N_{2i-1}, W_{2i-1} \xrightarrow{\mathrm{R}(s_i, o_i)} N_{2i}, W_{2i} \xrightarrow{\mathrm{W}(s_i, o_{i+1})} N_{2i+1}, W_{2i+1}$$

The above defines a sequence of read and write operations permitted by the policy starting from the given state. The sequence starts by a subject reading from the initial object o_1 and reflects the possibility that any subsequent write operation by that subject is possibly influenced by information in o_1. Any other subject that reads an object written to by s_1 may in turn be influenced by o_1 (whether intentionally or not), and hence any object they later write to may be influenced by confidential data in o_1. Clearly, there can be many such flows starting in a given state.

For an example of a flow, consider part of an information flow between o_2 and o_4 in Fig. 7. Beginning in state $\emptyset, \emptyset$, the flow in question can be enabled by the following small step transitions.

$$\begin{array}{ll}
\xrightarrow{\mathrm{R}(s_1,o_2)} & \{(s_1, o_2)\}, \emptyset \\
\xrightarrow{\mathrm{W}(s_1,o_3)} & \{(s_1, o_2), (s_1, o_3)\}, \{(s_1, o_3)\} \\
\xrightarrow{\mathrm{R}(s_2,o_3)} & \{(s_1, o_2), (s_1, o_3), (s_2, o_3)\}, \{(s_1, o_3)\} \qquad (\dagger) \\
\xrightarrow{\mathrm{R}(s_1,o_1)} & \{(s_1, o_1), (s_1, o_2), (s_2, o_2), (s_2, o_3)\}, \emptyset \\
\xrightarrow{\mathrm{W}(s_2,o_4)} & \{(s_1, o_1), (s_1, o_2), (s_1, o_3), (s_2, o_3), (s_2, o_4)\} \{(s_2, o_4)\}
\end{array}$$

In the above, the state marked with (†) corresponds to the left hand side of Fig 7. The read operation following that state, also appearing in the figure, is not active in the information flow under scrutiny, since it follows a read. It should be considered as part of a big step transition comprising the final two operations together, where only the second is part of this particular flow. That is, the final two operations above correspond to the big-step transition.

$$\begin{array}{l}
\{(s_1, o_2), (s_1, o_3), (s_2, o_3)\}, \{(s_1, o_3)\} \\
\xRightarrow{\mathrm{W}(s_2,o_4)} \{(s_1, o_1), (s_1, o_2), (s_1, o_3), (s_2, o_3), (s_2, o_4)\}, \{(s_2, o_4)\}
\end{array}$$

Notice that there is not a state of the system where all the read and write operations between o_2 and o_4 are simultaneously enabled. In contrast, the transitions in Fig.8 are not part of a flow from o_1 to o_3. This is because write access to o_3 is revoked before o_1 is read. These two examples help explaining why the rich notion adopted from Kessler is appropriate for modelling information flow.

B. Mechanising confidentiality via {log} and Coq

Now that we have the missing ingredients to define Theorem 4, we can proceed to mechanise the proof using {*log*} and Coq. The following recalls *starProp* from Sec. IV.

Definition 5. *For state N, W, we have $*(N, W)$ whenever, for all $(s, o) \in W$, s, o and N satisfy the *-property (Def. 2).*

We already mentioned that the above is proven to be invariant in {*log*}. To be precise, the following expresses what is mechanised in {*log*}.

Lemma 1. *If $N, W \xrightarrow{\alpha} N', W'$ then $*(N, W) \Rightarrow *(N', W')$. Furthermore, $N \subseteq N'$ and $\alpha = \mathrm{R}(s, o) \Rightarrow (s, o) \in N'$ and $\alpha = \mathrm{W}(s, o) \Rightarrow (s, o) \in W'$. Also, $*(\emptyset, \emptyset)$ holds.*

We then observe that big-step transitions also preserve the properties that we guaranteed in Lemma 1

Corollary 1. *If $N, W \xRightarrow{\alpha} N', W'$ then $*(N, W) \Rightarrow *(N', W')$. Furthermore, $N \subseteq N'$ and $\alpha = \mathrm{R}(s, o) \Rightarrow (s, o) \in N'$ and $\alpha = \mathrm{W}(s, o) \Rightarrow (s, o) \in W'$.*

Proof. Consider $N_0, W_0 \xRightarrow{\alpha_n} N_{n+1}, W_{n+1}$, and proceed by induction on the number of one-step transitions. In the base case, there is a one-step transition, and hence the result follows immediately from Lemma 1. Consider the inductive case where $N_0, W_0 \xRightarrow{\alpha_n} N_{n+1}, W_{n+1}$ and $N_{n+1}, W_{n+1} \xrightarrow{\alpha_{n+1}} N_{n+2}, W_{n+2}$. By the induction hypothesis, we have that $N_0 \subseteq N_{n+1}$ and $*(N_0, W_0) \Rightarrow *(N_{n+1}, W_{n+1})$. Furthermore, by Lemma 1, we have $N_{n+1} \subseteq N_{n+2}$ and $*(N_{n+1}, W_{n+1}) \Rightarrow *(N_{n+2}, W_{n+2})$, and $\alpha_{n+1} = \mathrm{R}(s, o) \Rightarrow (s, o) \in N_{n+2}$ and $\alpha = \mathrm{W}(s, o) \Rightarrow (s, o) \in W'$. Thus $N_0 \subseteq N_{n+2}$ and $*(N_0, W_0) \Rightarrow *(N_{n+2}, W_{n+2})$, as required. □

Having introduced these preliminaries, we can prove Theorem 4. As explained in Sec. III, the confidentiality property targeted by Brewer & Nash essentially says that along any flow, either the information flowing is sanitized or it stays within the same dataset. We use the Coq proof assistant to mechanise the proof of the following intermediate theorem, that proves that, in states where everything in W satisfies the *-property, the consequent of Theorem 4 holds. Hence, since we have a mechanised proof that the premise of Theorem 5 is an invariant in $\{log\}$, then Theorem 4 follows immediately from Lemma 1 and Theorem 5.

Theorem 5. *Consider any state N_1, W_1 such that $*(N_1, W_1)$. If o and o' are objects, then if $o \rightsquigarrow o'$ starts in N_1, W_1, then either $\mathrm{ds}(o) = Yo$ or $\mathrm{ds}(o) = \mathrm{ds}(o')$.*

Proof. Assume that $*(N_1, W_1)$ and also assume that o and o' are objects such that $o \rightsquigarrow o'$ starts in N_1, W_1.

By Definition 4, since $o \rightsquigarrow o'$, we have some n such that $s_1, \dots, s_n$; $o_1, \dots, o_{n+1}$; $N_1, \dots, N_{2n+1}$; and $W_1, \dots, W_{2n+1}$ such that $N_{2i-1}, W_{2i-1} \xRightarrow{\mathrm{R}(s_i, o_i)} N_{2i}, W_{2i} \xRightarrow{\mathrm{W}(s_i, o_{i+1})} N_{2i+1}, W_{2i+1}$, and $o = o_1$ and $o' = o_{n+1}$.

We then proceed by induction on n.

- Base case, $n = 0$. In this case $o \rightsquigarrow o$ and hence trivially $\mathrm{ds}(o) = \mathrm{ds}(o)$, as required.
- Induction hypothesis. For for $n = k$ we have if $o \rightsquigarrow o_{k+1}$ then either $\mathrm{ds}(o) = \mathrm{ds}(o_{k+1})$ or $\mathrm{ds}(o) = Yo$.
- Inductive case, $n = k + 1$.
  Notice that $o \rightsquigarrow o'$ can be decomposed into $o \rightsquigarrow o_{k+1} \rightsquigarrow o'$, where $o' = o_{k+2}$. In turn, $o \rightsquigarrow o_{k+1}$ is of length k so [by induction hypothesis] $\mathrm{ds}(o) = \mathrm{ds}(o_{k+1}) \vee \mathrm{ds}(o) = Yo$. If $\mathrm{ds}(o) = Yo$ we are done immediately, hence we consider next when $\mathrm{ds}(o) = \mathrm{ds}(o_{k+1})$.
  We now aim to establish that either $\mathrm{ds}(o_{k+1}) = Yo$ or $\mathrm{ds}(o_{k+1}) = \mathrm{ds}(o_{k+2})$ holds. Now, by Corollary 1, we have $*(N_{2i-1}, W_{2i-1}) \Rightarrow *(N_{2i}, W_{2i})$ and $*(N_{2i}, W_{2i}) \Rightarrow *(N_{2i+1}, W_{2i+1})$, for $1 \leq i \leq k+1$. Consequently, by transitivity repeatedly, we have $*(N_1, W_1) \Rightarrow *(N_{2k+3}, W_{2k+3})$ and, since we assumed that $*(N_1, W_1)$ holds, we have that $*(N_{2k+3}, W_{2k+3})$ holds. Furthermore, also by Corollary 1, we have $N_{2k+2} \subseteq N_{2k+3}$ and $(s_{k+1}, o_{k+1}) \in N_{2k+2}$ and $(s_{k+1}, o_{k+2}) \in W_{2k+3}$. Hence, since $N_{2k+2} \subseteq N_{2k+3}$ and $(s_{k+1}, o_{k+1}) \in N_{2k+2}$, we have $(s_{k+1}, o_{k+1}) \in N_{2k+3}$.
  Now, since $*(N_{2k+3}, W_{2k+3})$, by Def. 5, since we have $(s_{k+1}, o_{k+2}) \in W_{2k+3}$, it must be that s_{k+1}, o_{k+2}, and N_{2k+3} satisfy the *-property. Thus, by Def. 2, we have that $\mathrm{ds}(o_{k+1}) = \mathrm{ds}(o_{k+2})$ or $\mathrm{ds}(o_{k+1}) = Yo$ holds, since we have just established that $(s_{k+1}, o_{k+1}) \in N_{2k+3}$.
  Since we have just established that $\mathrm{ds}(o_{k+1}) = Yo$ or $\mathrm{ds}(o_{k+1}) = \mathrm{ds}(o_{k+2})$ holds and we are considering the case when $\mathrm{ds}(o) = \mathrm{ds}(o_{k+1})$, we have that either $\mathrm{ds}(o) = \mathrm{ds}(o_{k+2})$ or $\mathrm{ds}(o) = Yo$ holds, as required.

$\square$

The Coq proof—see the replication package accompanying this article [23]—is aligned with the proof shown above (e.g. it is based on induction over the length of the information flow). The reliance of the proof on Corollary 1 is achieved by assuming appropriate axioms in Coq. We cannot automate fully Theorem 5 in $\{log\}$ because $\{log\}$ cannot reason about arbitrary long sequences of operations (although it can reason about state invariants), nor about transitive closure, which is an alternative proof strategy.

VI. Minimum subjects to access all datasets, mechanized

In this section, we achieve full mechanisation of all theorems originally posed by Brewer & Nash. The missing proof, of Theorem 3, ensures that whenever all datasets in a CoIC are accessed then, there are at least as many subjects in the system as datasets. This is clearly a special case of a stronger theorem stating that, for any CoIC, the number of datasets that have been accessed in that CoIC is no greater than the total number of subjects in the system. The proof of this theorem can be automatically handled by $\{log\}$, by using a few tricks which we explain next.

Let C be the set of all possible CoICs and let $\mathcal{S}$ be the set of subjects in the system. Also, define the set of all datasets of CoIC X accessed in state N as follows.

$$D_X^N = \{Y \mid \exists (s, o) \in N : \mathrm{coi}(o) = X \wedge \mathrm{ds}(o) = Y\} \qquad (3)$$

The strengthening of the consequent of Theorem 3 mentioned above can be formulated as follows.

$$\forall X \in C : |D_X^N| \leq |\mathcal{S}| \qquad (4)$$

To see why proving that the above is an invariant establishes Theorem 3, consider X_v as defined in Sec. III. Observe that if, for all datasets $Y \in X_v$, there exists subject s and object o such that $(s, o) \in N$ and $\mathrm{ds}(o) = Y$, then we also have that $X_v = D_X^N$.

As we know from set theory, comparing the cardinality of sets, as in Eq (4) can be achieved by exhibiting a surjective partial function from $\mathcal{S}$ to D_X^N. We can construct such a surjective partial function as follows:

$$f_X^N = \{(s, Y) \mid \exists (s, o) \in N : \mathrm{coi}(o) = X \wedge \mathrm{ds}(o) = Y\} \qquad (5)$$

Since f_X^N is surjective, a given dataset Y can be accessed by more that one subject but a subject cannot access more than one dataset in a CoIC. Furthermore, since the range covers all datasets in the CoIC that have been accessed, there must be at least as many subjects as datasets being accessed in the CoIC. Thus it remains to check only the following, to ensure our construction is correct (ran denotes the co-domain of a relation).

$$\forall X \in C : \mathsf{pfun}(f_X^N) \wedge \mathsf{ran}(f_X^N) = D_X^N \qquad (6)$$

We use several tricks to achieve the effect of verifying Eq (6) in $\{log\}$. Firstly, a new state variable, *Sds*, is added to the model from in Sec. IV. *Sds* is a set of ordered pairs of the form (X, f) where $X \in C$ and f is a set of ordered pairs (s, Y) where s is a subject and Y a dataset. Second, the following invariant is added to the model to ensure that each f is a partial function.

$$\mathsf{invariant}(minSub).$$
$$minSub(Sds) \text{ :- } \forall (X, F) \in Sds : \mathsf{pfun}(F).$$

Thirdly, *Sds* is updated whenever a read access is requested, to ensure that f is kept in step with f_X^N above. For instance, in *spRead* the update is performed by conjoining the following (recall that S is the subject requesting access to an object whose label is (X, Y)):

$$\begin{aligned}
&\mathsf{dom}(Sds, A) \\
&\wedge (\quad X \in A \\
&\qquad \wedge \mathsf{ApplyTo}(Sds, X, R) \\
&\qquad \wedge \mathsf{oplus}(Sds, \{(X, \{(S, Y) \ / \ R\})\}, Sds') \\
&\quad \vee X \notin A \\
&\qquad \wedge Sds' = \{(X, \{(S, Y)\}) \ / \ Sds\})
\end{aligned}$$

That is, Sds is updated depending on whether X is already in Sds's domain or not. In the first case, (S, Y) is added to the image of X through Sds[8], whereas in the second case $(X, \{(S, Y)\})$ is added to Sds. In other words, Sds is updated in such a way that every time a subject s reads from a new object o whose security label is (X, Y), then (s, Y) is added to the image of X through Sds. Put it in other way, in any state, if X is a CoIC, then $Sds(X)$ is the set of pairs (s, Y) such that subject s is reading from some object o such that $L(o) = (X, Y)$.

Finally, two more invariants are included ensuring that N and Sds are always aligned. That is, the first invariant states that if (X, f) belongs to Sds and (S, Y) belongs to f, then there exists $(S, (O, \ell))$ in N such that $\ell = (X, Y)$. The $\{log\}$ code is the following.

$$\begin{aligned}
&\mathsf{invariant}(align_Sds_N). \\
&align_Sds_N(Sds, N) \text{ :-} \\
&\quad \forall (X, F) \in Sds; (S_1, Y) \in F : \\
&\qquad \exists (S_2, (O, (X_1, Y_1))) \in N : S_2 = S_1 \wedge X_1 = X \wedge Y = Y_1.
\end{aligned}$$

The second invariant (namely *align_N_Sds*) states the opposite inclusion—i.e., if an ordered pair is in N, then there's a corresponding ordered pair in Sds.

Discussion on scope: We have now fully interpreted and mechanised the original work of Brewer & Nash. It is natural, in the future, to consider more comprehensive indicators that conflicts-of-interest are avoided. For example, we could check that there is never a flow to a subject from two objects in different datasets but the same CoIC. This is stronger than Theorem 3, since we should prove that indirect flows from objects to subjects are also mitigated (Theorem 4 concerns flows from objects to objects). The scope of the current paper however is complete, since we aimed to clarify and mechanise the original Brewer-Nash model.

VII. Related and Future work: support for the policy maker

There are refinements of Brewer-Nash and related security policy models that can be analysed, some already mentioned in the introduction. This work can be seen as laying down a methodology that can be used to automate the analysis of such security policy models. Indeed, elements similar to our more explicit approach appear in Kessler [7], who treats access grants and operations separately. In related work, the Bell-LaPadula policy model has already been verified in $\{log\}$ [20]. In that work, VCs were manually generated. The presence of the VCG in the current paper not only automates a nontrivial task but, mainly, increases confidence in the correctness of the VCs and the model, by reducing the possibility of human error in the toolchain.

Real policies can become complex and we argue that the "policy maker" can benefit with efficient automated tools for analysing design decisions. Consider for example Fig. 9, where the right-hand side can only be expressed using the more general conflict-of-interest relation due to Lin [6]. Suppose a new policy model (not Brewer-Nash) permits a subject s_2 to write to o_2 while retaining read access to another dataset. In this case, (1) the conflicts-of-interests themselves are updated such that the dataset of o_1 absorbs the conflicts-of-interest of the dataset of o_3, and, (2) since that would create violation of the simple security property, read access to o_2 is revoked entirely for s_3. That is, access for one subject is revoked due to actions of another subject and furthermore the whole system becomes more restrictive, leading to conflict resolution questions. The analysis of information flow properties becomes trickier when read can be revoked since, Lemma 1, which assumes that read access monotonically increases, would be violated.

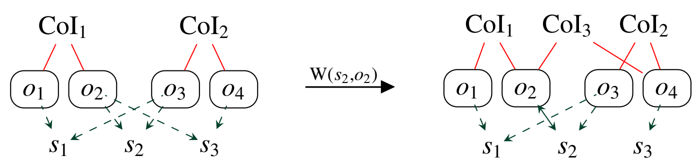

Fig. 9. Can a policy allow s_2 to insist on write access to o_2? Does this result in an updated conflict-of-interest relation and also read access of s_3 to o_2 being revoked?

In this work, we have extensively made use of $\{log\}$ to automatically verify properties and theorems. The same tool allows policy makers to simulate the behaviour of their policy model before attempting any serious proofs without any extra effort. In effect, the tool can be used to retrieve the post-state after having executed one of the specified operations (e.g. *rvkRead* or *readWrite*) given a particular pre-state. The replication package [23] includes guidelines to simulate some of the scenarios and examples included in the paper. The same tool's feature may also be used to discover the pre-state of an operation given the post-state (reverse simulation) or verify if a property holds in a particular given state.

Consider for example the pre-state shown in Fig. 9. If we want to verify whether *starProp* holds or not, we can ask $\{log\}$ to solve the following.

$$\begin{aligned}
&N = \{(s_1, (o_1, (c_1, d_1)), (s_1, (o_3, (c_2, d_3))), (s_2, (o_2, (c_1, d_2))), \\
&\qquad (s_2, (o_3, (c_2, d_3))), (s_3, (o_2, (c_1, d_2))), (s_3, (o_4, (c_2, d_4)))\} \\
&\wedge W = \emptyset \\
&\wedge starProp(yo, N, W).
\end{aligned}$$

[8] oplus is a $\{log\}$ constraint interpreted as B's *overriding* operator.

Values of N and W are returned by {*log*}, meaning *starProp* is satisfiable, otherwise the answer would have been no (unsat).

Furture work includes verifying the relationship between the implicit model at the beginning of Sec. II, and the explicit model in Fig. 3. Recall that the implicit model did not include the matrix W, and W was key for proving invariants establishing Lemma 1, in addition to making Brewer-Nash more implementable.

$$\frac{\forall o' : (s, o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{ds}(o') = Yo)}{N \xrightarrow{\mathrm{W}(s,o)} N}\ iW$$

$$\frac{\forall o' : (s, o') \in N \implies (\mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{ds}(o') = Yo)}{N \xrightarrow{\mathrm{W}(s,o)} N \cup \{(s, o)\}}\ iRW$$

$$\frac{\forall o' : (s, o') \in N \implies \mathrm{ds}(o') = \mathrm{ds}(o) \vee \mathrm{coi}(o) \neq \mathrm{coi}(o')}{N \xrightarrow{\mathrm{R}(s,o)} N \cup \{(s, o)\}}\ iR$$

Fig. 10. Implicit rules for Brewer-Nash policies, including sanitized data.

When enhanced with santized data, as shown in Fig. 10, the implicit and explicit models align in the sense that any operation in one is possible in the other. They are even bisimilar, an observation guaranteeing that results concerning information flow are preserved in the implicit model. We leave these formal comparisons as future work.

VIII. Conclusion

The theme of this agenda is to equip policy makers such that they may make bolder well-informed decisions regarding policies, preserving confidentiality constraints on information while potentially increasing access. How can such policy makers be sure that their policies preserve their intended information flow properties? More specifically, in this work, we argue that the widespread usage of Chinese Wall policies and high-stake consequences of policy failure, mean that we should not rely solely on the definitions and original proofs of Brewer & Nash and we should bring them up to the level of assurance given by modern mechanised tools. Indeed, we have mechanised in {*log*} invariants formulated in terms of the simple security rule and *-property (Theorem 1, Lemma 1 & Sec. IV); and also that the number of subjects in a system cannot be less than the number of datasets accessed in each conflict of interest class (Theorem 3 & Sec. VI). The steps of Theorem 4 mechanised in Coq are expressed in Theorem 5.

We have deliberately stuck closely to the original Brewer-Nash security policy model in this work. This is to remove any doubt that we verify anything other than the core model proposed by Brewer & Nash, and also because we believe that, even for that model, the operational semantics of access was left somewhat open to interpretation, as elaborated on in Sections II and III. Indeed, in Section II we have pointed out that our interpretation is not unique (e.g., if write access can be granted without read access, that opens up an initial phase where data can be pushed from subjects to multiple datasets independently of conflicts-of-interest, before read is granted within a single dataset, resulting in write being revoked elsewhere). Furthermore, the space of existing and future extensions of Brewer-Nash is large, as touched on in Section VII, and the systematic exploration of that space is open to creativity, where debates may be further substantiated by adapting the methodology employed in the current paper.

References

[1] D. F. Brewer and M. J. Nash, "The chinese wall security policy." in *IEEE symposium on security and privacy*, vol. 1989. Oakland, 1989, p. 206.

[2] S. N. Foley, "Building chinese walls in standard unixtm," *Computers & Security*, vol. 16, no. 6, pp. 551–563, 1997.

[3] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, and L. Van Doorn, "Building a mac-based security architecture for the xen open-source hypervisor," in *21st Annual Computer Security Applications Conference (ACSAC'05)*. IEEE, 2005, pp. 10–pp.

[4] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," *Draft MTR, The MITRE Corporation*, vol. 2, 1973.

[5] S. Hunt and D. Sands, "A quantale of information," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE, 2021, pp. 1–15.

[6] T. Y. Lin, "Chinese wall security policy-an aggressive model," in *1989 Fifth Annual Computer Security Applications Conference*. IEEE Computer Society, 1989, pp. 282–283.

[7] V. Kessler, "On the chinese wall model," in *Computer Security — ESORICS 92*, Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 41–54.

[8] A. Sharifi and M. V. Tripunitara, "Least-restrictive enforcement of the chinese wall security policy," in *Proceedings of the 18th acm symposium on access control models and technologies*, 2013, pp. 61–72.

[9] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems, 3rd Edition*. Wiley, 2020.

[10] M. Cristiá and G. Rossi, "Automated proof of Bell–LaPadula security properties," *Journal of Automated Reasoning*, vol. 65, no. 4, pp. 463–478, 2021.

[11] R. S. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9–19, 1993. [Online]. Available: https://doi.org/10.1109/2.241422

[12] G. Rossi, "{*log*}," http://www.clpset.unipr.it/setlog.Home.html, 2008, last access 2022.

[13] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi, "Sets and constraint logic programming," *ACM Trans. Program. Lang. Syst.*, vol. 22, no. 5, pp. 861–931, 2000.

[14] M. Cristiá and G. Rossi, "Solving quantifier-free first-order constraints over finite sets and binary relations," *J. Autom. Reason.*, vol. 64, no. 2, pp. 295–330, 2020. [Online]. Available: https://doi.org/10.1007/s10817-019-09520-4

[15] ——, "Automated reasoning with restricted intensional sets," *J. Autom. Reason.*, vol. 65, no. 6, pp. 809–890, 2021. [Online]. Available: https://doi.org/10.1007/s10817-021-09589-w

[16] ——, "Integrating cardinality constraints into constraint logic programming with sets," *Theory Pract. Log. Program.*, vol. 23, no. 2, pp. 468–502, 2023. [Online]. Available: https://doi.org/10.1017/S1471068421000521

[17] M. Cristiá and G. Rossi, "A decision procedure for a theory of finite sets with finite integer intervals," *ACM Trans. Comput. Logic*, sep 2023. [Online]. Available: https://doi.org/10.1145/3625230

[18] M. Cristiá and G. Rossi, "A set-theoretic decision procedure for quantifier-free, decidable languages extended with restricted quantifiers," *CoRR*, vol. abs/2208.03518, 2022, under consideration in Journal of Automated Reasoning. [Online]. Available: https://doi.org/10.48550/arXiv.2208.03518

[19] ——, "An automatically verified prototype of the Tokeneer ID station specification," *J. Autom. Reason.*, vol. 65, no. 8, pp. 1125–1151, 2021. [Online]. Available: https://doi.org/10.1007/s10817-021-09602-2

[20] ——, "Automated proof of Bell-LaPadula security properties," *J. Autom. Reason.*, vol. 65, no. 4, pp. 463–478, 2021. [Online]. Available: https://doi.org/10.1007/s10817-020-09577-6

[21] J.-R. Abrial, *The B-book: Assigning Programs to Meanings*. New York, NY, USA: Cambridge University Press, 1996.

[22] G. Rossi and M. Cristiá, “$\{log\}$ user's manual,” Dipartimento di Matematica, Universitá di Parma, Tech. Rep., 2020, https://www.clpset.unipr.it/SETLOG/setlog-man.pdf. [Online]. Available: http://https://www.clpset.unipr.it/SETLOG/setlog-man.pdf

[23] Anonymous, “The chinese wall security policy scrutinised,” Feb. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10611774