Rumpole: A Flexible Break-glass Access Control Model

Srdjan Marinovic
Department of Computing
Imperial College London, UK
srdjan@imperial.ac.uk

Robert Craven
Department of Computing
Imperial College London, UK
r.craven@imperial.ac.uk

Jiefei Ma
Department of Computing
Imperial College London, UK
j.ma@imperial.ac.uk

Naranker Dulay
Department of Computing
Imperial College London, UK
n.dulay@imperial.ac.uk

ABSTRACT

Access control operates under the assumption that it is possible to correctly encode and predict all subjects' needs and rights. However, in human-centric pervasive domains, such as health care, it is hard if not impossible to encode all emergencies and exceptions, but also to imagine *a priori* all the permissible requests. Break-glass is an approach that embodies the idea that under certain conditions it is possible for a subject to *break-the-glass* and explicitly override the denied request. Current break-glass models make this decision without considering and investigating what the reasons for issuing the denial are, and they have a fixed decision procedure to determine whether the override is permitted. Furthermore, they do not explicitly represent and reason over conflicting and missing information about subjects and the context; which in human-centric pervasive domains is a norm rather than an anomaly. This paper presents a novel break-glass model, Rumpole, that structures a break-glass policy by establishing why the access was denied. It uses Belnap's four-valued logic to represent conflicting and missing (unknown) information, allowing the policy to make a more informed decision when faced with missing or inconsistent knowledge. The model also provides a declarative query language that is used to specify an explicit break-glass decision procedure, rather than having an implicitly hard-coded one. This allows a policy writer to further condition and restrict when and how break-glass access is permitted.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security, Languages, Theory

Keywords: access control, break-glass, emergency management, policy

SACMAT'11, June 15–17, 2011, Innsbruck, Austria.

1. INTRODUCTION

Computer systems often employ a security policy that governs which subjects, under which conditions, are permitted to use resources. The policy is often enforced through the access control model that uses a preventive protection strategy. A crucial assumption underlying the access control system is that the encoded security policy is complete and can anticipate and precisely formulate which subjects ought to be permitted to access a resource.

However, in domains where it is not possible to completely encode or anticipate who should be granted access under which conditions, this assumption can result in obstructing people in their tasks, thus creating more risk than it tried to prevent. Risannen et al. [20] refer to these situations as "not machine encodable". For example, precise encoding of an emergency may never be complete, yet it is necessary to allow some access in precisely those situations. Conversely, it may be well understood what an emergency is but not who ought to be permitted. The common access control approach to these situations is not to grant the access, thus preserving the confidentially and integrity of resources, but at the cost of restricting availability. Even adopting an *open* access control policy, where a subject is permitted unless explicitly denied, does not alleviate the problem. For example, a subject could be denied because there is no emergency (but this is incompletely encoded), or simply because it was not anticipated that the subject needs the access.

Break-glass access control has recently been proposed [19, 2] as a way to supplement traditional access control models in order to deal with these situations. The core idea of the approach is to allow the subject to override the access control decision, whilst imposing *obligatory* actions on the subject or the system itself. The main difference between standalone and break-glass policies is that the former encode the conditions for permitting access, while the latter encode the conditions under which overrides are permitted. The benefits of allowing overriding are established in pervasive computing domains such as emergency management [2] and health-care [11]—but even more traditional domains such as business applications are adopting break-glass concepts such as the Virsa Firefighter for SAP software [1].

An essential aspect of override permissions is comprehending why the denial was first issued. Clearly there is a difference between subjects' being explicitly *denied* and not being explicitly *permitted*. Equally, a subject may be typically permitted (e.g. has the right role or credentials) but a particular

access will violate integrity constraints such as separation-of-duty. Current break-glass models do not attempt to represent precisely why the request was denied. This leaves the break-glass policy to be defined only in terms of contextual conditions without being able to constrain the break-glass overrides based on knowledge of the denial's causes.

Modality conflicts (where a subject is permitted and denied at the same time) commonly occur when making an access control decision and the access control model must employ a conflict-resolution strategy. But a *conflict* value can be useful to communicate to the break-glass policy, as it clearly shows that there is evidence to support both a permission and a denial. In a distributed system there could be many access decision points and their decisions could disagree (without any of them having internal conflict). However, when combining these decisions it is clear that the *conflict* value is needed to accurately determine the overall decision. There are also situations where the access control simply has too little information and it is unknown whether the subject is permitted or denied. In these situations some form of Closed World Assumption (CWA) is adopted, where the unknown value is assumed to be false, but *unknown* would be more useful. The need to evaluate a decision as *unknown* appears quite naturally in distributed scenarios where decision points can be unreachable. In pervasive systems these problems are even more acute as the decisions are based on sensor data (to establish the context) which can be conflicting or missing. Current break-glass approaches do not offer a framework to represent these different truth and knowledge levels and no way to define how such values are to be combined in order to reach a decision.

It is of vital importance to be able to constrain the overrides by imposing non-contextual conditions such as limit of overrides per subject or target, how much incomplete knowledge is allowed, the least number of available access decision points, etc. These fine-grained integrity constraints are not currently considered by break-glass models. Furthermore, current models have an implicit decision procedures which dictates how the break-glass decision is reached. A fixed decision procedure tells the PDP which break-glass policies, in which order, they are to be considered, thus the policy writer is forced to construct the break-glass policy according to this fixed decision procedure.

This paper presents a novel break-glass model, Rumpole, to be used as an extension of access control models. Rumpole's contributions are three. (1) It introduces notions of subjects' *competences* and *empowerments* to gain more insight into the causes for the access denial, (2) It uses Belnap's four-valued logic [6] to represent conflicting and incomplete information, and this logic underlies the semantics of rules used to encode how these facts are combined to evaluate how much is known about the subject and whether he may override. (3) The model has a declarative query language to specify a break-glass decision procedure, this allows a policy writer to condition and constrain the break-glass access permissions in a fine-grained manner by embedding integrity constraints into the decision procedure.

2. OVERVIEW OF RUMPOLE MODEL

This section provides an informal account of Rumpole model where a high-level break-glass policy is encoded using the following concepts:

- *Competences* – Encode whether a subject has necessary abilities to access the resource without causing harm to the resource or the system.
- *Empowerments* – Encode whether the necessary contextual conditions are met so that the access will not cause harm to the resource or the system.
- *Break-glass Rules* – Encode whether a subject is permitted or denied to override an access control denial based on what is known about his competences, empowerments and obligations that he has accepted or violated.
- *Resolution Query* – Encodes how the rules, and under what conditions, are consulted in order to reach an override decision.

2.1 Integration with Access Control Points

As indicated in the Introduction, Rumpole is not an extension of a particular access control model, such as presented in [12] for RBAC. This creates a clear separation of concerns between the two models and more importantly it allows the break-glass decision point to be able to consult different, and possibly distributed, access control knowledge bases. Figure 1 captures how this separation between the access control decision points (ACPs) and the break-glass decision point is organised. *PDP* represents a security policy decision point

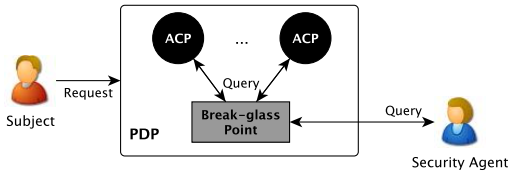

Figure 1: Integration of the break-glass decision point within a PDP

which consists of two types of decision points: access control points and a break-glass point. When the subject requests a particular access, the PDP will consult its access control point(s) to determine whether the access can be given. How their decisions are combined to reach a decision is independent of the break-glass policy. If the request is denied, the PDP will forward the request to the break-glass point to see if this decision can be overridden by the break-glass policy encoded through Rumpole. If the override can be given, the subject will be notified about the possible obligations that he will have to fulfil and asked to confirm that he accepts them. This *confirmation* interaction is not part of the model and is left to be enforced by the PDP implementation.

The *Query* abstraction represents the interaction between the break-glass point and an individual ACP, when a break-glass point attempts to determine why the denial was issued (we assume that the access control point can answer queries about the state of its knowledge base). Naturally, the question arises of what the query should contain, or how the denial should be represented. In this paper we propose that the denial is depicted by the concepts of competences and empowerments.

However, the break-glass point need not be reliant only on a set of access control points. We use the term *Security Agent* to refer to either human or computational agents who can testify for or against subjects' attributes. The reason why we explicitly separate an ACP and a security agent is

that not all security agents are qualified to make an access control decision. This is especially prevalent in pervasive systems such as health care, where nurses can often supply security relevant information on the subjects or even make certain local security decisions.

2.2 Competence and Empowerment

Fundamentally, an access control model attempts to determine whether the subject requesting the access is permitted, given the context of the request. As put forward in Barker [4], all subjects are organised into categories (which could be roles, domains and so forth), and permissions hold between a subject's category and a requested action on a resource. Determining the *who* part of the permission means determining if the subject's category is sufficient for allowing the access, and the category inclusion can be contextually constrained.

On the other hand, the *contextual* conditions describe integrity constraints which are not part of the conditions that determine which category the subject falls into or which permissions a category has. Typical examples of these conditions represent separation-of-duty constraints, where a subject can *typically* access a resource but owing to his current open sessions or previous accesses cannot be permitted. Similarly, a particular access may not be allowed after working hours. Even though this condition could be encoded through the definition of a subject's categories, it is more naturally expressed as an integrity constraint.

It follows that the fundamental understanding of why the denial was issued lies in determining whether a subject's category (if it was assigned to any) has the necessary permissions and also whether such permissions broke any of the contextual constraints. We characterise these two aspects as: ***(1) competence*** and ***(2) empowerment***.

The notion of competence subsumes the notion of *belonging to a permitted category* and attempts to capture whether the subject possesses abilities necessary for accessing the resource in a way that would not cause any undesired harm. For example, belonging to a *nurse* category implicitly carries the notion that the subject has the skills to perform certain nursing tasks on patients and should be given access to appropriate patient data. Competence could also be established by a set of certificates and credentials. However, the notion of competence does not have to be exclusively linked to a particular role, domain or credentials. Those attributes are used as *evidence* for a subject's competence. This can augmented by additional evidence that is not directly linked to a particular access control model. So, for example, a break-glass policy might say that after a certain number of reviewed and approved overrides a subject can be considered as competent.

The notion of *empowerment* subsumes these of resource-specific and contextually-specific integrity constraints, and captures whether the access should take place without considering whether some particular subject is competent to make it. For example, a subject, in a typical separation-of-duty constraint, may not be permitted to execute two workflow conflicting tasks, and this information could be used as evidence to support or diminish his empowerment. But much as is the case with competence, empowerment does not have to depend exclusively on integrity constraints.

From the given discussions, it follows that a minimal understanding of why a denial was issued can be obtained by using the access control knowledge base to provide evidence to support or disprove subjects' competences and empowerments. Thus having the necessary credentials can be used as positive evidence towards establishing competence, but breaking the integrity constraints is negative evidence for an empowerment. This kind of intimate understanding of the access control decision gives us a fine-grained way to structure break-glass policies.

Notice that apart from being *true* and *false*, both competences and empowerments can also be *unknown*. In other words, if the subject does not posses the necessary credentials it may be more appropriate to say that it is unknown whether it is competent rather than adopting the Closed World Assumption (CWA) and automatically assuming that it is not. On the other hand, not belonging to a particular category may be deemed as sufficient to establish that the subject is not competent. Clearly we need the additional *unknown* value to specify how the access control knowledge base is to be interpreted and used. Furthermore, if a particular distributed access control point, or a security agent, cannot be reached, then their knowledge should be evidently characterised as *unknown*.

Similarly, a subject may be deemed competent by one access control point and not competent by another. The appropriate value for the competence is therefore *conflict*. However, even locally some evidence may support a subject's competence (such as having the credentials) but other evidence such as broken obligations may go against it, resulting in the *conflict* value. A better understanding of whether a subject is competent and empowered can be achieved by expanding the truth-value space, rather than attempting to represent these attributes using the classical truth values of "true" and "false". Since there is a clear distinction between the four truth values of "true", "false", "conflict" and "unknown", they can be used to specify a more precise break-glass policy.

2.3 Break-glass Rules

Given what and how much is known about a subject's competences and empowerments, we need to specify how their truth values are to be combined in order to establish whether an override is permitted or not. The override decision can be further conditioned by the obligations that a subject has to agree upon or conditioned by contextual situations surrounding the request. These break-glass rules are split into two types: (1) permit rules and (2) deny rules.

Permit rules specify how much is known about whether an override is permitted. The meaning of the *true* value is that it is known that the subject is permitted. The *false* value indicates that it is known that the subject is not permitted. The *unknown* value tells the PDP that it cannot be established whether the override is permitted. Finally the *conflict* value indicates that there is evidence to support both cases. Hence, expanding the truth-value space is not only beneficial for reasoning about subjects' competences and empowerments but also for allowing a clear statement of what and how much is known about their override permissions as well.

Often it is needed to state more explicitly that a subject is denied an override, much like the integrity constraints are used in access control policies. We refer to these rules as deny rules. Strictly speaking it is not necessary to have deny rules: one may attempt to embed these constraints

into permit rules and make sure that they are evaluated as false. This will, however, create complex rules that are hard to specify and manage. Not only can deny rules be used to create a less complex policy, they can also be used to give different semantic weight to denial constraints when considering them in conjunction with the permit rules. For example, in certain cases the *deny* may take precedence over the *permit* decision and similarly not permitted may be taken as stronger than not denied. This is addressed in more detail in Section 7.

2.4 Break-glass Resolution Query

A break-glass resolution query specifies how the break-glass decision point infers a binary override decision which either permits an override or not.

The resolution query specifies how to combine knowledge about whether the subject is permitted or denied. But it also allows us to define how much weight is given to each decision and how much knowledge is deemed sufficient in order to make a conclusive decision. For example, for some resources only full knowledge that the request is not explicitly denied may be required, but for more sensitive resources the request would have to be permitted as well. The resolution query thus gives a fine-grained way to specify whether to override access control based on understanding how much is known, how much is unknown, or in conflict.

Informally: the break-glass rules establish how much is known about whether the request is permitted or denied, and the resolution query then specifies how such information is used to instruct the PDP what to do.

3. BELNAP'S $\mathcal{FOUR}$ LOGIC

In [6] Belnap introduced a logic based on four different truth values: the *classical* values t (true) and f (false), and two additional ones, $\bot$, intuitively denoting lack of information (no knowledge), and $\top$, denoting inconsistency or conflict of information (*over*-knowledge). It is commonly referred to as the logic $\mathcal{FOUR}$. The main notion behind

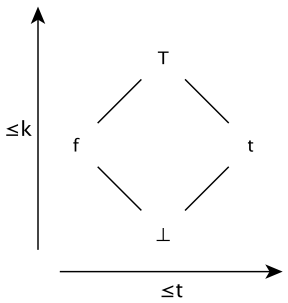

Figure 2: The Logic $\mathcal{FOUR}$ [6]

these truth values is to convey how much is known regarding the truth of a sentence, when the evidence is gathered from possibly multiple sources.

The truth values have two natural orderings (Figure 2). The first is the standard logical partial order $\leq_t$, which reflects differences in the *measure of truth* that every value represents. f is the least element, t is the greatest one, while $\top$ and $\bot$ are the intermediate values that are incomparable w.r.t. $\leq_t$. As Figure 2 indicates, these four values form a lattice with respect to the $\leq_t$ ordering. The finite meet and join of this lattice, $\wedge$ and $\vee$ respectively, correspond to the classical truth operators. Negation is represented through an order reversing involution unary operator $\neg$, for which $\top = \neg\top$, $\bot = \neg\bot$. For the values t and f, the truth operators $\wedge$ and $\vee$ behave as in classical logic.

The second order, $\leq_k$, reflects the differences in the amount of knowledge or information that each truth value exhibits. Again the four values form a lattice such that $\top$ is the maximal element, $\bot$ is the minimal, and t and f are incomparable w.r.t. $\leq_k$. Fitting [13, 14] introduced symbols $\otimes$ and $\oplus$ to denote respectively the finite meet and join operations. The $\otimes$ can be seen as giving the most amount of information that the truth values can agree on, while the $\otimes$ can be seen as giving the most amount of information that can be derived. So for example $f \oplus t \equiv \top$, which fits into our intuitive notion that having supporting evidence for both the truth and falsity will give rise to a conflict. Similarly $\top \oplus A \equiv \top$ for all A, essentially says that once a conflict is established for a certain value adding any more information cannot change this. On the other hand $f \otimes t \equiv \bot$, which says that there is no information that is agreed upon. All binary operators are monotonic w.r.t both orderings (if $x_1 \leq_* y_1$ and $x_2 \leq_* y2$ then x_1 *op* $x_2 \leq_* y_1$ *op* y_2) but the negation operator is only monotonic w.r.t $\leq_k$ ordering.

The previously introduced idea that the break-glass policy (encoded in Rumpole) needs to differentiate whether a certain subject's property or a system state is known to be true or false, or unknown, or conflicting can be mapped to the Belnap truth values. So these truth values cover the sought-after truth space, while the Belnap operators provide the necessary connectives to describe how to combine various pieces of evidence.

4. LANGUAGE $\mathcal{L}_E$

The language $\mathcal{L}_E$ is used by Rumpole to encode rules that define how competences, empowerments and override permissions are determined. Every rule can be seen as a piece of evidence contributing towards this evaluation. The language $\mathcal{L}_E$ is a multi-sorted function-free language used for specifying correlations between Belnap predicates. It is defined over the $\mathcal{FOUR}$ truth values and we assume the usual logic programming notions of a term, literal, and a predicate. An atomic formula, or simply an atom, is an expression $p(t_1, ..., t_n)$ where p is a predicate of arity n and t_i is a term.

An interpretation I of a set of rules P is a function that to every ground atom from the Herbrand base of P assigns a Belnap truth value. Formulae of $\mathcal{L}_E$ are expressions build up from atoms using the introduced Belnap operators: $\neg$, $\wedge$, $\vee$, $\oplus$, $\otimes$, and constants $\{\top_{\mathbf{4}}, \bot_{\mathbf{4}}, t_{\mathbf{4}}, f_{\mathbf{4}}\}$.

DEFINITION 1. *The interpretation I is extended pointwise to Belnap formulae in the following manner:*

- $I(\top_4) = \top$ *and similarly for other constants as well.*
- $I(\neg\psi) = \neg I(\psi)$ *where ψ is an $\mathcal{L}_E$ formula.*
- $I(\psi \wedge \psi') = I(\psi) \wedge I(\psi')$ *where ψ and ψ' are $\mathcal{L}_E$ formulae, and similarly for operators: $\vee$, $\otimes$, and $\oplus$.*

$\mathcal{L}_E$ rules are split into two types, the unconditional and the conditional ones. In order to make the presentation of the semantics clearer, we shall first consider the unconditional

rules and the semantics of the specification that only contains these rules; after that we shall expand the semantics to include the second type of rules as well.

DEFINITION 2. *An unconditional rule is an expression $A \Leftarrow \psi$, where A is an atom and ψ is a formula.*

An unconditional rule with a grounded atom as its head and a constant as its body is referred to as a fact.

The unconditional rule should intuitively be thought of as a rule of testimony that says that there is evidence for the atom A to have at minimum the truth value of ψ: in other words $I(A) \geq_k I(\psi)$ for every rule. Just as in the case for choosing an interpretation for the semantics of the normal logic programs, where the interpretation may be required to be minimal w.r.t. the number of true atoms, we would also like the interpretation I to be minimal in the amount of knowledge that it assigns. But it should also be supported in the sense that it assigns only as much knowledge as needed to satisfy all the rules. In order to construct this minimal and supported interpretation I, we will use a fixpoint operator T_P which maps an interpretation I onto interpretation I' in such a way that it applies the evidence rules over I to derive values for I'. We keep applying this operator in order to reach a fixpoint interpretation which we will show to be the minimal and supported interpretation that we are looking for.

The set of unconditional rules is first *grounded* and then with this grounded set of rules P a fixpoint operator T_P is associated, defined as:

$$T_P(I)(A) = \begin{cases} I(\psi_1 \oplus ... \oplus \psi_n) & \forall A \text{ where } A \Leftarrow \psi_i \in P \\ \bot & \neg\exists A \text{ where } A \Leftarrow \psi_i \in P \end{cases}$$

T_P maps an interpretation I onto interpretation I', such that for all A it holds that $I'(A) = T_P(I)(A)$. This defined operator is a modification of the operator that Fitting introduced in [13] where the ground atom A could appear in the head of exactly one ground rule. As these rules are used to formulate pieces of evidence for a particular attribute or property, they should be combined using $\oplus$ operator. The reason for this is that the intuitive meaning of the knowledge base's interpretation is that which assigns the least amount of knowledge such that all the rules are taken into account.

This way of combining the rules is in contrast to normal logic programs where the operator $\vee$ is used, and to further illustrate why we have chosen $\oplus$ instead of $\vee$ let us consider the set of rules P, $\{a \Leftarrow \top_{\mathbf{4}},\ a \Leftarrow \bot_{\mathbf{4}}\}$. If the $\vee$ operator is used, $I_P(a) = t$, where the $\oplus$ will result in $I_P(a) = \top$. It seems counterintuitive that if we have evidence for a to suggest that it is both in conflict and unknown then the interpretation should convey that a is true.

Since all the Belnap operators are monotonic with respect to $\leq_k$ (and by structural induction all formulae ψ made out of these operators), it follows that the T_P is a *monotonic* operator, thus for any two interpretations it holds that:

$$I_1 \leq_k I_2 \Rightarrow T_P(I_1) \leq_k T_P(I_2)$$

Clearly this monotonicity does not extend the truth-ordering $\leq_t$ as negation is a non-monotonic operator with respect to truth-ordering. Furthermore Fitting has also established that the operator T_P is *(chain) continuous*, and for any chain of interpretations $I_1 \leq_k I_2 \leq ...$ it holds that:

$$T_P(\bigoplus_i I_i) = \bigoplus_i T_P(I_i)$$

By the Knaster-Tarski theorem the monotonic and continuous operator T_P [13, 14] has the least fixpoint $T_P{\uparrow}^\omega$ which can be constructed through:

$$I_P^0 \stackrel{def}{=} I_\bot$$
$$I_P^{n+1} \stackrel{def}{=} T_P(I_P^n)$$
$$\ldots$$
$$T_P{\uparrow}^\omega \stackrel{def}{=} \bigoplus_{\alpha<\omega} I_P^\alpha$$

where $I_\bot$ represents an interpretation which assigns $\bot$ to every atom from P's Herbrand base. We follow Fitting and take this least fixpoint, $T_P{\uparrow}^\omega$ as a canonical interpretation for P's declarative semantics.

Given T_P's definition and its properties, T_P can be seen as accumulating knowledge by progressively using the given rules as evidence to establish the least amount of knowledge such that the head of every rule has as little knowledge as possible and still be $\geq_k$ than its $I(\psi)$. And the least fixpoint is *minimal* and *supported* as it requires no more information in order for its rules' heads to be $\geq_k$ than their bodies. Thus if the value of any A is lowered in the least fixpoint, some rule will have $A \not\geq_k I(\psi)$, meaning that this piece of evidence has effectively not been taken into account. If a set of $\mathcal{L}_E$ rules are taken as evidential rules for encoding competences, empowerments and break-glass rules, then the least fixpoint combines all the evidence for a particular attribute in such a way to maximise, as much as it has to, the amount of knowledge that can be established about these attributes.

Notice that the fixpoint is non-monotonic with respect to the truth ordering: adding more facts or rules to P may turn the truth value of some of its atoms from t to f or vice-versa.

As mentioned at the beginning of this subsection the T_P operator can be used over the set P of unconditional rules, and the reason why we refer to these rules as *unconditional* is that the T_P operator considers them unconditionally as pieces of evidence, since in every iteration every rule is used. But this presupposes that each piece of evidence should be used to establish the Belnap value of a rule's head. At first this may seem right, after all this is exactly how the normal logic program's clauses are used. But let us consider the following example: *A nurse is competent to do any action on any target at any time.* The policy writer may write the following rule:

$$competent(Sub, Act, Tar, T) \Leftarrow role(Sub, nurse)$$

This rule is deceptively simple and it can give rise to some potentially unwanted consequences. For example when f is assigned to $role(Sub, nurse)$, the Sub will be considered as not competent and similarly when $I(role(Sub, nurse)) = \top$, the fixpoint will tell us that the Sub has both evidence for and against her competence. Clearly this encoding is wrong since the intuitive notion of the example is to say that if the Sub is nurse, it is competent but it does not address what to conclude when the Sub is not a nurse.

The solution to this is to use the *role* condition as the rule's *applicability* condition; if it is true then other conditions are evaluated, otherwise the rule is skipped. In normal logic programs when clauses are used as low-level policies, there is no need to separate the conditions as they are conjoined with the $\wedge$ operator and they can entail the $true$ value only when all conditions are $true$. This is no longer the case

with the rules in Belnap logic: more values can be entailed and it must be clearly stated when these values should be entailed. To express these conditions we propose the following *applicability* operator:

$$I(\psi \textbf{ if } \phi) = \begin{cases} I(\psi) & \text{if } I(\phi) = t \\ \bot & \text{otherwise} \end{cases}$$

using which a condition rule is defined as:

DEFINITION 3. *A conditional rule of $\mathcal{L}_E$ is an expression of the form $A \Leftarrow \psi$ **if** ϕ.*

The intuition behind a conditional rule is that its formula ψ contributes to the value of the atom A if and only if $I(\phi) = t$, when this is the case we refer to the rule as being applicable. Unfortunately the applicability operator is non-monotonic with respect to $\leq_k$. This has two immediate consequences: (1) it cannot be expressed with the available operators (thus it cannot be expressed in Fitting's language [13]), and (2) it is no longer possible to guarantee that the least fixpoint $T_P\uparrow^\omega$ can be constructed. Clearly not having the *applicability* operator can limit the extent to which the policy language can be used; it is likely to be essential for expressing certain policies.

DEFINITION 4. *A set of conditional $\mathcal{L}_E$ rules P is hierarchically stratified in n strata as $P_1 \cup ... \cup P_n = P$, when the predicates in rules' heads in one stratum do not appear as heads in other strata, the predicates in ψ contain only those from the same stratum or a lower one, and the predicates in ϕ contain only predicates from the lower strata.*

Unconditional rules can always be treated as conditional rules by having $t_{\mathbf{4}}$ as ϕ.

DEFINITION 5. *For a stratified set of $\mathcal{L}_E$ rules an iterated fixpoint interpretation I_P is defined as I_n:*

$$I_1 = T_{P_1}\uparrow^\omega$$
$$I_2 = T_{P_2 \cup I_1}\uparrow^\omega$$
$$\dots$$
$$I_n = T_{P_n \cup I_{n-1}}\uparrow^\omega$$

An interpretation I can be represented by a set of facts, where for every I's atom there is a conditional rule that has that atom as a head and the body has only a Belnap constant that corresponds to atom's value (given by I). The expression $P_{i+i} \cup I_i$ merges P_{i+1}'s rules with I_i's facts to create a knowledge base over which T_P is applied. We have to do this since T_P is not a progressive operator, and hence it cannot be guaranteed that for any starting interpretation I there exists the least fixpoint $T_P\uparrow^\omega(I)$.

PROPOSITION 1. *An iterated stratified interpretation I_P of a set of hierarchically stratified rules P is minimal (with respect to $\leq_k$ ordering) and supported.*

PROOF. The stratum P_1 contains only unconditional rules and thus I_1 is supported and a minimal interpretation. Adding I_1's facts to P_2 will essentially disable some conditional rules whose RHS of the applicability operator is not equal to t and this cannot change during the fixpoint construction. The immediate effect of this is that all the remaining rules can be treated as unconditional rules and T_{P_i} is thus monotonic and continuous. Hence I_2 is also supported and a minimal interpretation since the value of atoms in I_2 will not be changed by any higher strata. Therefore by induction the last interpretation I_n will also be supported and minimal. □

In summary $\mathcal{L}_E$ represents a novel extension of the bilattice logic programming language introduced in [13] by adding a k-nonmonotonic operator **if** and defining a new stratified semantics for the fixpoint operator.

5. ENCODING A BREAK-GLASS POLICY

Rumpole encodes a break-glass policy in two parts:

(1) ***Evidential rules*** – These are encoded as $\mathcal{L}_E$ rules and are used to define how various pieces of information, that either attest or disprove subjects' and contextual attributes and conditions, are combined to establish how much is known about competences and empowerments. The $\mathcal{L}_E$ rules are also used to encode the break-glass rules that establish how much is known about whether the override is permitted or denied. The semantics of these rules is given by constructing the minimal interpretation for the rule set, as described in the previous section.

(2) ***Resolution query*** – The evidential rules construct an *evidence* base (defined by the constructed interpretation) that tells the PDP how much truth and knowledge has been established by them. Now, the PDP needs to be instructed what to do based on this evidential information. A resolution query is used for this purpose as it encodes how to combine the *permit* and *deny* information (and potentially other contextual information) in order to allow or deny the override. In order to be able to encode a fine-grained resolution query, we need to be able to constrain how much knowledge is needed for a particular predicate and which predicates are to be given more weight. This kind of expressivity is not directly encodable through the $\mathcal{L}_E$ rules. Accordingly we have designed a query language designed for constructing fine-grained queries over a Belnap interpretation.

This strategy of separating the knowledge base from the actual decision is inspired by the similar approach used in SecPAL [5].

5.1 Pre-defined Sorts and Predicate Sets

Evidential rules of any high-level break-glass policy have to use at least the following sorts. (1) A finite sort of subjects *Subject*; the variable *Sub* will be used (possibly with superscripts and subscripts) to represent members of this sort. (2) A finite sort of actions *Action*; the variable *Act* will be used (possibly with superscripts and subscripts) to represent members of this sort. (3) A finite sort of targets *Target*; the variable Tar will be used (possibly with superscripts and subscripts) to represent members from this sort.

The predicates used in the evidential policies are split into three disjoint sets $\mathcal{L}_E^{Core}$, $\mathcal{L}_E^{Obl}$, and $\mathcal{L}_E^{aux}$. The set $\mathcal{L}_E^{Core}$ contains the predicates *permit* (args $Subject \times Target \times Action$), *deny* (args $Subject \times Target \times Action$), *competent* (args $Subject \times Target \times Action$), and *empowered* (args $Subject \times Target \times Action$). These are used to represent competences and empowerments as well as the break-glass rules. The set $\mathcal{L}_E^{Obl}$ contains the following predicates:

1. *agreedObl* (args $Subject \times Target \times Action$) – used to denote whether the subject has agreed to perform the requested obligatory action on the designated target.
2. *violatedObl* (args $Subject \times Target \times Action$) – used to indicate that the specified obligation has been violated by the subject.
3. *fulfilledObl* (args $Subject \times Target \times Action$) – used to

indicate that the specified obligation has been fulfilled by the subject.

4. *activeObl* (args $Subject \times Target \times Action$) – used to denote that the given obligation is still active.

The set $\mathcal{L}_E^{aux}$ holds additional *auxiliary* predicates used for representing domain and access control policy specific properties.

6. EVIDENTIAL RULES

6.1 Competences and Empowerments

Rules defining competences and empowerments are used as pieces of evidence to contribute towards establishing subjects' competence or empowerment.

DEFINITION 6. *A competence/empowerment rule is a conditional rule that has a competent or empowered atom as its head and body atoms from* $\mathcal{L}_E^{Core} \cup \mathcal{L}_E^{Obl} \cup \mathcal{L}_E^{aux}$

The interpretation of Belnap truth values for these predicates is as follows:

- $[competent/empowered](Sub, Tar, Act) = t$ – It is known that Sub is competent/empowered to perform Act on Tar.
- $[competent/empowered](Sub, Tar, Act) = f$ – It is known that Sub is not competent/empowered to perform Act on Tar.
- $[competent/empowered](Sub, Tar, Act) = \bot$ – It is not known whether Sub is competent/empowered or not to perform Act on Tar.
- $[competent/empowered](Sub, Tar, Act) = \top$ – There is conflict between whether the Sub can be considered as competent/empowered to perform Act on Tar.

To illustrate how competence may be specified let us consider the following example: *A student is competent to assist a patient when the student's supervising nurse assigned that student to the patient.* One simple attempt could be:

$$competent(Sub, Patient, assist) \Leftarrow assigned(nurse, Sub, Patient) \wedge student(Sub)$$

However, this states that the Sub is known not to be competent, i.e. $I(competent(Sub, Patient, assist) = f$, whenever the Sub does not hold the *student* role, i.e. $I(student(Sub)) = f$. This can result in a quite different decision over the override request from the intended, since the role atom contributes directly with its truth value towards establishing the competence's truth value. But formulating the rule as:

$$competent(Sub, Patient, assist) \Leftarrow assigned(nurse, Sub, Patient) \textbf{ if } student(Sub)$$

is more appropriate as it is used only when the Sub holds the *student* role and otherwise the rule is not used.

Apart from specifying explicit competences and empowerment, these rules should be used to extract fine-grained reasons why the denial was issued in the first place. To illustrate this further we shall use the influential FAF access control model by Jajodia et al. [16]. The FAF model allows both positive and negative authorisations (*dercando* predicate) to be specified as well as integrity constraints (*error* predicate). As mentioned, the notion of competence relates to authorisation based on the subject's inclusion in a particular role, or his place in a more general hierarchical domain. On the other hand, empowerments try to capture, not whether the subject has the necessary role, but whether such access can take place given the context. We can capture this analysis with the following rules:

$$competent(Sub, Tar, Act) \Leftarrow dercando(Sub, Tar, +Act) \oplus \neg dercando(Sub, Tar, -Act)$$
$$empowered(Sub, Tar, Act) \Leftarrow \neg error(Sub, Tar, Act)$$

where $dercando(Sub, Tar, +Act)$ is a positive authorisation and $dercando(Sub, Tar, -Act)$ is a negative authorisation. Thus, the competence is gauged by combining the information about whether a positive and/or a negative authorisations were derivable. This leaves the *empowered* predicate to be used as a query about whether the integrity (i.e. contextual) constraints have been broken, in other words the *when* component of the access control decision. Thus these constraints can be used to tell the PDP whether restrictions, such as *separation-of-duty*, were broken. Other access control models such as GTRBAC [17] can be queried in the similar fashion. Depending on the complexity of an access control model additional $\mathcal{L}_E^{aux}$ predicates may be required to correctly capture competences and empowerments.

6.2 Break-glass Rules

Break-glass rules are represented as evidential rules defining how much is known about whether a subject is permitted or denied to override an access control denial.

DEFINITION 7. *A break-glass rule is a conditional rule that has permit or deny atom as its head and body atoms from the set* $\mathcal{L}_E^{Core} \cup \mathcal{L}_E^{aux} \cup \mathcal{L}_E^{Obl}$.

Intuitively the *permit* predicate is used to provide evidence to support the override whereas the *deny* policy represents the evidence to support the denial of the override. Their meaning is as follows:

- $[permit/deny](Sub, Tar, Act) = t$ – It is known that Sub is permitted/denied to override the access denial for Act on Tar.
- $[permit/deny](Sub, Tar, Act) = f$ – It is known that Sub is not permitted/denied to override the access denial for Act on Tar.
- $[permit/deny](Sub, Tar, Act) = \bot$ – It is not known whether Sub is permitted/denied to override the access denial for Act on Tar.
- $[permit/deny](Sub, Tar, Act) = \top$ – There is conflict of evidence between whether the Sub is permitted/denied the override.

For example: *a subject is permitted to override and append to a file only when it is competent to read it and has agreed to provide the reason for this.* This example can be captured in the following way:

$$permit(Sub, file, append) \Leftarrow competent(Sub, file, read) \textbf{ if } agreedObl(Sub, log, giveReason)$$

Notice that when the Sub is not competent this rule will result in the interpretation saying that Sub is not permitted

which corresponds to the *only* condition in the wording of the policy. A slightly different example may say that an override is permitted for any access if the subject is known to be competent or if there is conflicting information about his competence, and he has not broken any obligations:

$$permit(Sub, Tar, Act) \Leftarrow competent(Sub, Tar, Act) \otimes t_4$$
$$\textbf{if } \neg violatedObl(Sub, Tar_2, Act_2)$$

The difference between this example and the previous one is that this example does not use competence as a way to dispute the override permission.

However, as has been discussed, we may need to provide some safeguards to constrain overrides and not expose resources to higher risks. The following examples capture some of these concerns: *A subject is not allowed to override when he exceeds the override limit*, and *during night shifts no overriding can take place if the subject is not empowered*:

$$deny(Sub, Tar, Act) \Leftarrow t_4$$
$$\textbf{if } overrideCount(Sub, N) \wedge exceededLimit(Sub, N)$$
$$deny(Sub, Tar, Act) \Leftarrow \neg empowered(Sub, Tar, Act)$$
$$\textbf{if } currentTime(T) \wedge nightShift(T)$$

These constraints are used to prevent potential abuse and also to limit overrides when needed. For example, during night shifts, there are fewer doctors present and thus potential emergencies caused by inappropriate overrides need to be kept to a minimum. These policies are used to encode evidence about a particular override request, and they are not supposed to be used as instructions for the PDP. Thus simply having encoded the break-glass policy is not enough to let the PDP make the decision over a certain request.

6.3 Evidence base

Previous subsections have shown how $\mathcal{L}_E$'s rules are used to encode evidence to represent a subject's attributes and to encode how much is known about whether a request is permitted or denied.

Definition 8. *An evidence base EB of a high-level break-glass policy is a set of locally stratified (see Def. 4) $\mathcal{L}_E$ rules, where all rules that have permit, deny, competent, empowered are as in Definitions 6 and 7.*

Proposition 2. *An evidence base has a minimal (with respect to $\leq_k$ ordering) supported interpretation.*

Proof. Follows directly from Proposition 1. □

We take the iterated stratified fixpoint of the evidence base, I_{EB}, as its intended semantic interpretation.

7. BREAK-GLASS RESOLUTION QUERY

Given a break-glass policy and its interpretation I_{EB}, Rumpole needs to enforce an override decision for a request (s, t, a) where $s \in Subject$, $t \in Target$, and $a \in Action$. Before we define how the decision is reached, we first define an access control query over the I_{EB} of an encoded break-glass policy.

Definition 9. *A break-glass resolution query Ω is defined as:*

$$\Omega ::= \phi \mid \Omega \wedge \Omega \mid \Omega \vee \Omega \mid \Omega \sqsupset_t \Omega \mid \Omega \sqsupset_f \Omega$$
$$\phi ::= b \mid p(x_1, ..., x_n) \mid \neg\phi \mid \phi \ op \ \phi$$

where $p(x_1, ...x_n)$ is a ground atom, $b \in \{t, f, \top, \bot\}$, and $op \in \{\wedge, \vee, \oplus, \otimes, \textbf{\textit{if}}\}$.

Definition 10. *A break-glass resolution query Ω is satisfied by an I_{EB} for a request (s, t, a), written as $I_{EB} \models \Omega$, as specified by the structural induction (where $* \in \{t, k\}$ and $** \in \{t, f\}$):*

- $I_{EB} \models \phi_1$ *op* ϕ_2 iff $I_{EB}(\phi_1)$ *op* $I_{EB}(\phi_2)$, *where op* $\in \{=, \neq, \leq_*, \not\leq_*\}$.
- $I_{EB} \models \Omega_1 \wedge \Omega_2$ iff $I_{EB} \models \Omega_1$ and $I_{EB} \models \Omega_2$
- $I_{EB} \models \Omega_1 \vee \Omega_2$ iff $I_{EB} \models \Omega_1$ or $I_{EB} \models \Omega_2$
- $I_{EB} \models \Omega_1 \sqsupset_{**} \Omega_2$ iff $I_{EB} \models \Omega_1 \sqsupset_{**} I_{EB} \models \Omega_2$

We have introduced the $\sqsupset_{[t/f]}$ operator as part of the query. This operator will be used to hierarchically structure subqueries within the resolution query. Its semantic evaluation over the I_{EB} is defined as:

$$I_{EB} \models \Omega_1 \ \sqsupset_t I_{EB} \models \Omega_2 \stackrel{def}{=} \begin{cases} true & \text{if } I_{EB} \models \Omega_1 \\ I_{EB} \models \Omega_2 & \text{otherwise} \end{cases}$$

$$I_{EB} \models \Omega_1 \ \sqsupset_f I_{EB} \models \Omega_2 \stackrel{def}{=} \begin{cases} false & \text{if } I_{EB} \models \Omega_1 \\ I_{EB} \models \Omega_2 & \text{otherwise} \end{cases}$$

Informally, the operator $\sqsupset_t$ says if the LHS formula is *true* then the whole expression is *true*, otherwise the truth value is determined by the RHS formula. The operator $\sqsupset_f$ says if the LHS is *true* then the whole expression is *false*, otherwise the truth value is determined by the RHS formula.

A PDP can only make a decision on whether to allow an override if it has a query to evaluate on a knowledge base. Therefore we define an encoded break-glass policy as:

Definition 11. *An encoded break-glass policy is a tuple $\langle I_{EB}, \Omega \rangle$, where I_{EB} is the evidence base of the break-glass policy and Ω is the break-glass resolution query.*

Definition 12. *A PDP will allow an override for the request (s, t, a) according to the encoded break-glass policy $\langle I_{EB}, \Omega \rangle$, iff $I_{EB} \models \Omega$ for the given s, t and a.*

The query can return only the two classical truth values *true* and *false*, and as the query is a propositional sentence it is decidable. To illustrate how a simple query can be constructed consider the following example:

$$\Omega_{conservative} \stackrel{def}{=} (permit(s, t, a) = t \wedge deny(s, t, a) = f)$$

where s, t and a are placeholders for the corresponding values grounded by the request. This can be considered as a conservative query as it allows an override only when it is known that the override is permitted and when it is known that it is not denied. We refer to it as a conservative query because it requires the knowledge to be fully established as known in both cases. Thus if a permission atom is unknown or in conflict the override will be denied.

A more tolerant query may allow an override as long as there is some evidence to support the override and no evidence to imply the denial in the following fashion:

$$\Omega_{tolerant} \stackrel{def}{=} (permit(s, t, a) \geq_t \top) \wedge (deny(s, t, a) \leq_t \bot)$$

The last example underlines that an essential part of a resolution query is stating how to treat conflicting and unknown

	permit	*deny*
Strong resolution conflict	t	t
Weak resolution conflict	$\top, \bot$	t
	t	$\top, \bot$
Strong resolution gap	f	f
Weak resolution gap	f	$\top, \bot$
	$\top, \bot$	f
Strong resolution incompleteness	$\bot$	$\bot$
Weak resolution incompleteness	$\top$	$\bot$
	$\bot$	$\top$

Table 1: Conflicts, Gaps and Incompletenesses

information in order to make an override decision. Clearly modality conflicts can occur when it is known that the request is both permitted and denied. However the expanded truth-space introduces additional situations which can occur when evaluating break-glass rules which are summarised in Table 1. The resolution gap represents a situation where it is known (fully or partially) that the request is neither permitted or denied. Resolution incompleteness occurs when it is not possible to reach a clear understanding of whether the request is permitted or denied. Thus, a resolution query can distinguish between these situations and potentially make a more cautious decision based on available or unavailable knowledge. For example in case of conflicting or unknown evaluations, the policy may decide to trust the subject and allow the request if the subject agrees to leave a fingerprint:

$$\Omega_{lax} \stackrel{def}{=} (permit(s,t,a) \geq_t \top) \wedge (deny(s,t,a) \leq_t \bot) \sqsupset_t$$
$$(deny(s,t,a) <_t t) \wedge agreedObl(s, fingerPrint, take)$$

Intuitively, the Ω_{lax} query allows an override if there is some evidence that it is permitted and it is known that it is not denied. If the first sub-query does not allow it, then the user can leave a fingerprint and still be allowed to override provided that at least it is known he is not denied.

However, the Ω_{lax} query may be seen as too risky when the override involves sensitive resources. The query can easily be restricted to address this issue in the following manner:

$$\Omega_{restricted\text{-}lax} \stackrel{def}{=}$$
$$(permit(s,t,a) \geq_t \top) \wedge (deny(s,t,a) \leq_t \bot) \sqsupset_t$$
$$(sensitive(t) \geq_t \top) \sqsupset_f$$
$$(deny(s,t,a) \leq_t t) \wedge agreedObl(s, fingerPrint, give)$$

Here the query evaluation is stopped if the target is a sensitive resource and the user has not been allowed an override before reaching this point. This results in the query only using the risky override for non-sensitive resources.

8. RELATED WORK

One of the earliest arguments for a break-glass concept was formulated by Povey [19], where it was argued that there will always be an expressiveness gap between what can be encoded and what the needs of an organisation are. The author introduced partially-formed transactions, whose effects can be *rolled-back*, and the core idea was to allow users to perform these transactions even if they do not have the permission but are willing to acknowledge that they are aware of this fact. Risannen et al. [20] have similarly argued that all requests cannot be anticipated and that many conditions are not completely encodable. Their model provides the predicate *can* which permits the requestor to override a denied decision, but this has to be authorised and the model determines who this should be. Rumpole does not address this issue. The Break-glass concept has been introduced into the Role-based Access Control (RBAC) model within a medical information system [12, 11] where a user is permitted to override any access as long as he acknowledges the override. Brucker et al. [8] presented a generic break-glass model where subjects are permitted to override specific access control permissions. The access control policy consists of a partially ordered set of permissions; to each permission, override permissions are attached. These are enabled by activating pre-defined emergency levels. Ardagna et al. [3] present a break-glass model where the policies are separated into different categories starting with the access control policies, emergency policies and a break-glass policy. Unless the access is explicitly denied, it can be obtained by either finding an applicable emergency policy with obligations or, if that is not successful, the override is granted if the system is in some emergency state and the supervisor can be notified about the override.

These break-glass models hard-code the break-glass resolution procedure into the model itself rather than, as with Rumpole, expressing it as a declarative rule over the causes for the denied access, which can be varied from policy to policy. Thus a policy writer has to frame a break-policy according to a particular model's break-glass procedure which in turn can limit the expressivity of the intended break-glass policy. For example in Risannen et al.'s work, the policy cannot explicitly define override constraints, while Brucker et al.'s work relies on a predefined set of break-glass policies and emergencies and unless the emergency is activated, no override is permitted. In Ardagna et al.'s work explicit access control denials can never be overridden, and the override depends on correctly encoding and identifying emergency situations. In contrast our work attempts to structure the break-glass policy based on understanding which access control conditions were broken, thus avoiding explicit idea of an "emergency" state; furthermore Rumpole takes advantage of reasoning over unknown and contradictory knowledge to permit an override even if it is unknown or contradictory that there is an emergency. Similarly, in situations where if it is known that the subject does not have an override permission, one can still be given if the subject has not broken any prior obligations and if the override will not break any integrity constraints. This declarative way of expressing a break-glass policy empowers the policy writer to represent a more expressive break-glass policy that is not tied into any particular break-glass procedure. Rumpole can also encode the implicit break-glass resolution procedures present in all surveyed break-glass models.

Obligations in access control have an established presence [18, 7, 15]. These models augment an access control policy with obligations and are thus aimed at formulating extended access control policies rather than break-glass policies. The obligation models in these languages are more expressive than the obligations in this paper and we plan to explore how the break-glass obligations can be made more expressive by using these proposed solutions.

Bruns and Huth presented a policy language PBel [21, 9] which is also based on Belnap logic. PBel has a number of operators that are used to construct a policy as a composition of sub-policies. PBel's operators are syntactic

constructs over Belnap operators and an additional operator $a \supset b$. This additional operator can be encoded as: $(b \textbf{ if } a \otimes t_{\boldsymbol{4}}) \oplus (t_{\boldsymbol{4}} \textbf{ if } \neg(a \oplus f_{\boldsymbol{4}}))$. Hence $\mathcal{L}_E$ could express PBel's operators if multiple **if** operators were used within a rule, which would not alter the presented stratified semantics in any way. PBel does not support policy variables, and it cannot express recursive policy definitions.

9. CONCLUSION AND FUTURE WORK

This paper presents a novel break-glass model, Rumpole, that takes a different approach to structuring a break-glass policy which is based on identifying causes for the denial, rather than on a set of emergencies or explicit override permissions. Rumpole uses a logic programming language defined over Belnap logic to explicitly reason about unknown and conflicting information. This reasoning over different amounts of knowledge, coupled with explicit override constraints, provides a flexible and generic break-glass model.

Our *proof-of-concept* implementation implements the T_P operator over grounded rules. Although the construction of the evidence base is polynomial with respect to the size of the rule base, this is not scalable as the grounding process can generate an exponential number of grounded instances. Also the *bottom-up* construction creates the whole evidence base whenever a new fact is inserted. To overcome these issues, we have a procedure which translates $\mathcal{L}_e$'s rules into a stratified Datalog program [10]. Proofs of soundness and completeness are forthcoming. The Datalog encoding can then take advantage of top-down polynomial resolution procedures to infer atoms' truth values. As mentioned in the paper, the current query resolution does not support variables in the query which is something that we are currently expanding the query semantics with. The queries will be then translated into a Datalog fragment as well.

Having multiple **if** operators in an $\mathcal{L}_E$ rule allows it to have any PBel [9] policy-composition expression as its body. Similarly we are looking into translating certain fragments of D-Algebra (specifically its P-Interpretation) [22] into a set of $\mathcal{L}_E$ rules. Thus we attempt to investigate to what extent $\mathcal{L}_E$ can be used as a general access control policy language.

10. ACKNOWLEDGMENTS

We thank the reviewers for their comments and suggestions. Marinovic and Dulay's research was supported by EU FP7 research grant 213339 (ALLOW). Craven and Ma's research was supported by the International Technology Alliance, sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

11. REFERENCES

[1] Virsa firefighter for sap, http://www.sap.com/uk/solutions/solutionextensions/virsa/index.epx.

[2] Break-glass: An approach to granting emergency access to healthcare systems. *White Paper, Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC)*, 2004.

[3] C. A. Ardagna, S. D. C. di Vimercati, S. Foresti, T. W. Grandison, S. Jajodia, and P. Samarati. Access control for smarter healthcare using policy spaces. *Computers & Security*, 29(8):848 – 858, 2010.

[4] S. Barker. The next 700 access control models or a unifying meta-model? In *SACMAT '09*, pages 187–196, 2009.

[5] M. Y. Becker, C. Fournet, and A. D. Gordon. Secpal: Design and semantics of a decentralized authorization language. *J. Comput. Secur.*, 18:619–665, 2010.

[6] N. D. Belnap. A useful four-valued logic. *Modern Uses of Multiple-Valued Logics*, pages 8–37, 1977.

[7] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy management and security applications. In *VLDB '02*, pages 502–513, 2002.

[8] A. D. Brucker and H. Petritsch. Extending access control models with break-glass. In *SACMAT '09*, pages 197–206, 2009.

[9] G. Bruns and M. Huth. Access-control policies via belnap logic: Effective and efficient composition and analysis. In *CSF '08*, pages 163–176, June 2008.

[10] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, Mar 1989.

[11] A. Ferreira, D. Chadwick, P. Farinha, R. Correia, G. Zao, R. Chilro, and L. Antunes. How to securely break into rbac: The btg-rbac model. In *ACSAC '09. Annual*, pages 23 –31, 2009.

[12] A. Ferreira, R. Cruz-Correia, L. Antunes, P. Farinha, E. Oliveira-Palhares, D. W. Chadwick, and A. Costa-Pereira. How to break access control in a controlled manner. In *CBMS '06*, pages 847–854, 2006.

[13] M. Fitting. Bilattices and the semantics of logic programming. *J. Log. Program.*, 11(2):91–116, 1991.

[14] M. Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1-2):25 – 51, 2002.

[15] K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *CCS '06*, pages 134–143, 2006.

[16] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[17] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. on Knowl. and Data Eng.*, 17:4–23, January 2005.

[18] Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control policies and privacy policies. In *SACMAT '08*, pages 133–142, 2008.

[19] D. Povey. Optimistic security: a new access control paradigm. In *NSPW '99*, pages 40–45, 2000.

[20] E. Rissanen, B. S. Firozabadi, and M. J. Sergot. Discretionary overriding of access control in the privilege calculus. In *Formal Aspects in Security and Trust*, pages 219–232, 2004.

[21] G. Bruns, D. S. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In *FMSE '07*, pages 12–21, 2007.

[22] Q. Ni, E. Bertino, and J. Lobo. D-algebra for composing access control policy decisions. In *ASIACCS '09*, pages 298–309, 2009.