Tool (col.) / Feature support (row)	Custom Attribute Types	Relational Rules	Static Verification	Dynamic Verification	t-way Testing	XACML Support
Alloy [22]	○	-	●	-	-	●
ACPT [23]	●	○	●	●	●	●
Margrave [24]	-	○	●	-	-	◐
SPIN [25]	-	-	●	-	-	-
ACCOn [26]	-	-	●	-	-	-

Figure 8. A brief comparison of the model checking tools we considered. A more complete treatment can be found in the work of Aqib and Shaikh [27].

simultaneous grasp of the richness and constraints of the language expressing the authorization policy. As a result, these authorization policies are either too expressive or not expressive enough. Defining BTG requirements within authorization policies adds more complexity, which can easily result in misconfigurations and faulty policies, introducing serious vulnerabilities. Therefore, rigorous verification and validation through systematic testing are required to ensure the security properties are satisfied, the access control model is expressed correctly in the authorization policy, multiple policies enforced simultaneously are consistent, and single policies are self-consistent. Two policies can be called inconsistent if they are both applicable to a specific access request and yet return different decisions, e.g. one returns "allow" while the other returns "deny".

5.1. Tool Selection

To formally verify and test authorization policies for our BTG framework, we explored a series of access control policy test tools. Many were either not compatible with our model or missing properties/features that we required for thorough testing. Tools which we would consider good candidates for validating our work should support model-based verification in addition to properties enumerated below. The features marked **bold** are required while others are simply helpful.

1. **Different types of attributes**: XACML has four categories for attributes by default, subject, action, resource, and environment categories. These categories are supported by available XACML implementations such as WSO2 Balana [28]. We are looking for support for the default and additional categories, including customized contextual categories of attributes with discrete and continuous values, since the types of attributes used in practice (e.g. at a medical facility) are not limited to these four categories. Additional categories can be added as needed.

2. **Rules with relational expressions**: A simple XACML policy such as in Figure 5 can contain conditions composed of relational expressions (written as logical expressions).

3. **Static and dynamic verification**: Although we currently only use static verification, both are useful in detection and resolution of inconsistency and incompleteness in policies. In a realistic deployment, we envision a static policy check before they are enacted, and continuous dynamic checking to detect problems after deployment.

4. t-way combination tests: A policy developer can easily end up with hundreds of access control polices even for a relatively small size organization. t-way combinatorial testing is useful for generating smaller, more manageable test suites and reducing testing costs [29].

5. Native support for XACML: Our access control policies are written in ALFA and then translated into XACML for reasons of compatibility and portability. Therefore, the tool needs to support importing, processing, and exporting standard XACML policies.

Our initial search resulted in more than 12 tools or approaches. We filtered these tools by methods used and only focused on the approaches that were based on model checking. This reduced the list to 5. Our next step in filtering was to check for the tools that use or support XACML for policy specifications, but filtering by XACML would have left us with a very limited number of tools. Therefore, we also looked into tools that did not support XACML but some other policy specification language. A brief summary of the tools or approaches based on model checking is given in Table 8. We refer the readers to Aqib and Shaikh [27] for a detailed survey of verification and validation tools and approaches for access control policies.

The Access Control Policy Testing (ACPT) tool [23, 30], developed by the National Institute of Science and Technology (NIST) comes closest to fulfilling our requirements above. It can be used not only to compose and generate access control polices, but also to verify and test these policies, supporting both static and dynamic verification. The tool uses the Symbolic Model

Verification (SMV) model checker [31] for property checking and Automated Combinatorial Testing for Software (ACTS) [32] for test suite generation. In addition, ACPT allows for policies to be either merged or combined prior to verification or testing.

5.2. Checking Policy Consistency

The verification of safety requirements stated as properties can assure only the logical integrity of the policy rules against the specific safety requirements. This is a heuristic approach – although we use model checking, the model cannot provide complete coverage because of temporal logic within the policy, and so it may not cover all possible values of all rules (or all conditions in the rules).

Figure 9 shows the list of requirements for our authorization policy (Figure 5) in the form of high-level security properties along with the status for each of the properties returned by the tool after verifying it. The result confirms that all properties hold.[4] A policy is shown correct by checking a set of properties that the system should satisfy against. We verify our model against the below specified properties:

- Request to update or view *flowRate* by *physician* or *nurse* should always be granted when *BTG* is set to true, as shown in lines 1 and 2. *flowRate* is a medical resource which belongs to group BTG-allow and should be granted access if a clinician requests it during an active BTG session. These properties verify the rule *allowEmergencyAccess* from the policy set in Figure 5.

- The property in line 3 ensures that *physician* should never be granted access to "BTG-restricted" resources when *BTG* is set to true. This property particularly verifies the condition in the rule *allowEmergencyAccess*, which restricts emergency access (BTG) to resources that belong to "BTG-restricted" group. Line 4 further generalizes this check by replacing *physician* with *anyone*. A resource-specific version of this property (denying access for *physician* to modify *auth-policy* – a "BTG-restricted" resource – while BTG is enabled) is given in line 5.

- Line 6 checks that *sys-admin* should be allowed to update *auth-policy* when the system is not in *BTG* state. Line 7, on the other hand, checks that if BTG is active, then the same request should be denied. Only *sys-admin* is allowed to update or view *auth-policy* and since *auth-policy* belongs to "BTG-restricted" group, requesting access to it by *sys-admin* should be granted only if the system is in normal state. These properties verify the rule *allowModifyAuthPolicy*.

- The property in line 8 is used to verify that a request for setting *btg* (e.g. BTG access) flag by anyone should be granted unless the system is already in BTG state. This requirement is defined in the rule *allowBtg*.

- The property in line 9 verifies the BTG requirement in the rule *allowEmergencyAccess* – anyone with role *visitor*[5] requesting *write* access to *flowRate* should be granted if *BTG* is set to true and the resource *flowRate* belongs to *BTG-allow* group for the user. Recall the rule *allowEmergencyAccess* does not require a specific role (e.g. see physician and nurse in lines 1 and 2, respectively).

```
1 spec AG (((((role = "physician" & resource = "flowRate") & group = "BTG
      -allow") & DefaultAction = "write") & BTG = "True") -> decision =
      Permit) is true
2 spec AG ((((role = "nurse" & resource = "flowRate") & group = "BTG
      -allow") & BTG = "True") -> decision = Permit) is true
3 spec AG (((role = "physician" & group = "BTG-restricted") & BTG = "True"
      ) -> decision = Deny) is true
4 spec AG ((group = "BTG-restricted" & BTG = "True") -> decision = Deny)
      is true
5 spec AG (((((role = "physician" & resource = "auth_policy") & group = "
      BTG-restricted") & DefaultAction = "write") & BTG = "True") ->
      decision = Deny) is true
6 spec AG (((((role = "sys_admin" & resource = "auth_policy") & group = "
      BTG-restricted") & DefaultAction = "write") & BTG = "False") ->
      decision = Permit) is true
7 spec AG (((((role = "sys_admin" & resource = "auth_policy") & group = "
      BTG-restricted") & DefaultAction = "write") & BTG = "True") ->
      decision = Deny) is true
8 spec AG (((resource = "btg" & group = "normal") & BTG = "False") ->
      decision = Permit) is true
9 spec AG (((((role = " visitor " & resource = "flowRate") & group = "BTG
      -allow") & DefaultAction = "write") & BTG = "True") -> decision =
      Permit) is true
```

Figure 9. Results of ACPT verification of policy consistency **as specified by the facility**, reformatted for readability. An inconsistency within a facility's policy corpus will cause at least one of the specifications to evaluate as "false" and the tool will provide a counterexample.

5.3. Testing Results

We use ACPT to validate a **slightly modified version** of the BTG policy shown in Figure 5, and verify that our model meets safety requirements for emergency override. For example, the safety requirement in line 4 of Figure 9 (expressed in temporal logic) formalizes the rule `allowEmergencyAccess` in the example BTG policy in Figure 5, and is checked for any violations verified using the model checker.

[4] To avoid confusion we use BTG (in capital letters) as a flag indicating status of the BTG state of the system and btg (in small letters) as a resource in policies.

[5] A visitor could be a visiting doctor from a different healthcare facility who is holding temporary security credentials.

```
1  spec AG (((((role = "nurse" & resource = "flowRate") & group = "BTG
       -allow") & DefaultAction = "write") & BTG = "True") -> decision =
       Permit) is false as demonstrated by the following execution
       sequence:
2  Trace Description: Counterexample-
3       > ABAC_ polSetBtgFlag.decision = Deny-
4       > ABAC_ polSetFlowRate.decision = Permit
```

Figure 10. Policy inconsistency shown with a counterexample, reformatted for readability

The reason for verifying a slightly modified policy in the previous paragraph can now be revealed: Figure 10 shows **faults in the seemingly trivial policy**. The issues are **inconsistency** and **incompleteness**. The `authPolicySet` was initially defined with the `denyOverrides` policy combining algorithm, which favors "deny" decisions, and was considered the safest for our example. During verification, however, the property on line 9 from Figure 9 returned false with a counterexample demonstrating **incompleteness** via a "not applicable" decision, since no matching rule was found. To resolve this, we added default deny rules to the polices to ensure that in cases where there is not a policy matching an access request, a deny decision is still enforced. This resulted in other properties failing due to conflicting decisions (**inconsistency**). For example, the property on line 2 ensures that a request for a resource in the "btg-allow" group should always be allowed provided that the system is in a BTG state (recall this property is defined in `allowEmergencyAccess` rule in Figure 5). During verification, `polSetFlowRate` returned "allow" and `polSetBtgFlag` returned "deny", resulting in a "deny" decision being returned by the combined policy set (`authPolicySet`) because of the combining algorithm `denyOverrides`, as shown in Figure 10. Replacing `denyOverrides` with `permitOverrides` resolved the inconsistency and the final verification result is shown in Figure 9. Conflicting and missing rules are hard to detect without a policy verification and evaluation engine, i.e. a clinician or policymaker may not foresee the consequences of a policy combination until an unexpected result occurs (likely during a medical procedure), and it may be difficult to determine the reason for the unexpected result after-the-fact, making the policies difficult to fix once deployed.

When a new resource is added to the list of available resources, it should be "flagged" for administrative review, and an access control policy for the identified resource should be written. Normally a resource without a written access control policy is considered non-existing and the policy decision point (PDP) will return an indeterminate or not applicable result if the resource is requested for access. In our design and implementation of a sample BTG policy we

```
1  (resource="btg")&(BTG="False")&(role="sys_admin")&(group="BTG
       -allow")->Permit
2  (resource="auth_policy")&(BTG="True")&(role="sys_admin")&(group
       ="normal")->Deny
3  (resource="flowRate")&(BTG="False")&(role="sys_admin")&(group="
       BTG-restricted")->Deny
4  (resource="btg")&(BTG="False")&(role="physician")&(group="
       normal")->Permit
5  (resource="auth_policy")&(BTG="True")&(role="physician")&(group
       ="BTG-restricted")->Deny
6  (resource="flowRate")&(BTG="True")&(role="physician")&(group="
       BTG-allow")->Permit
7  (resource="btg")&(BTG="True")&(role="visitor")&(group="BTG
       -restricted")->Deny
8  (resource="auth_policy")&(BTG="False")&(role="visitor")&(group=
       "BTG-allow")->Deny
9  (resource="flowRate")&(BTG="False")&(role="visitor")&(group="
       normal")->Deny
10 (resource="btg")&(BTG="False")&(role="nurse")&(group="BTG-allow
       ")->Permit
11 (resource="flowRate")&(BTG="True")&(role="nurse")&(group="BTG
       -restricted")->Deny
12 (resource="auth_policy")&(BTG="True")&(role="nurse")&(group="
       normal")->Deny
```

Figure 11. Results of ACPT's auto-generated heuristic testing, reformatted for readability and with metadata header removed

added default deny rules to avoid not applicable and indeterminate results in situations such as those. Also, if a well written access control policy (e.g. meta-policy) is written for resources such as medical devices or apps that share some properties, then a new policy is not necessary to be written. For example, by adding a new medical device that is either part of a new setup with identical configurations to an existing device or a replacement for an existing device in the system there needs to be no further actions to be taken except for approving the installation by an administrator.

Mistakes such as misconfigurations or faulty policies can remain undetected even after some thorough verification and testing. Our approach is designed to make "misassigning" a BTG-restricted resource difficult, unless one also adds or modifies the resource-specific policy in addition to the overall BTG policy. "Mis-assinging" a BTG-restricted resource, which is similar to not including a resource in a blacklist, can introduce system vulnerabilities. We built on the concept of a policy fault model presented in [33, 34] to check for misassigned resources. We introduced new resources and misassigned them to the list of resources other than BTG-restricted resources. Testing results for these resources were either "indeterminate" or "not applicable". By re-running the tests and adding default deny rules to these policies, the results changed to "deny". However, access to resources, for which we also added new policies or modified their existing policies, the authorization engine returned decisions as expected. Therefore, the testing confirmed the need for a policy modification step for already misassigned resources before they can allow unwanted access.

In Figure 11, we perform conformance testing to validate compliance of implementation of the policy model. The tool automatically generates test cases from the model using combinatorial array generation technology [32], evaluates test requests against the policy, and returns the testing results – which can be used for identifying conflicts, inconsistencies, and other type of faults in the given policies.

Properties like those in Figure 9 can be written for an identified fault and verified using ACPT. In Figure 11 each of the test cases generated for the BTG policy in Figure 5 consists of certain attributes (resource, role, group, BTG) and a final decision:

- Users should be granted access to the resource *btg* (e.g. initiating a BTG session) if the BTG flag is set to false, confirmed by the test cases 1, 4, 7, and 10.
- A user should not be granted permission to access the resource *auth_policy* when the system is in a BTG state, validated by test cases 2, 5, and 12.
- Similarly, test cases 5, 7, and 11 validate that access to any resource in the group *BTG-restricted* will be denied during BTG.
- Test cases 3, 8, and 9 are used to validate access restrictions during the normal operating state (e.g. non-clinicians – system administrators – should not be able to set medication dispensing rates).
- Finally, in test case 6, we check that a user will be granted access to any resource in group *BTG-allow*, other than *btg* and *auth_policy*, only if the system is in a BTG state.[6]

Note that **not all possible cases are covered due to the use of t-way combinatorial testing**. We were able to perform 2-way, 3-way and 4-way combinatorial testing on the given policy set. Due to space limitations, we only report on 2-way testing results.

6. Conclusion, Limitations, and Future Work

Current access controls override mechanisms in medical systems allow for uncontrolled access when needed, which may leave systems open to misuse and unnecessarily compromise patients' privacy, resulting in irreversible damage. The controlled emergency access model in our work allows for a flexible emergency access control override while ensuring system safety- and security-critical resources are protected even during the Break the Glass state by managing system state and BTG obligations as specified in a formally verified and validated authorization policy. Furthermore, we show how the authorization architecture allows for the system to return to a known safe state and reduce or eliminate the need for manual audits when returning from a controlled BTG session. Finally, we formally show that our example policy is expressed correctly and the policy specifications are satisfied.

While our work makes some headway towards flexible emergency access (BTG) within existing authorization frameworks, there are several important limitations, derived mostly from our limited access to non-PHI healthcare data. Given the difficulty in locating healthcare providers willing to participate in sharing data on their facility-specific policies regarding access control override mechanisms, authorization policies, and log data, we could only achieve a limited understanding of how our proposed model would fit in a real healthcare setting. An in-depth qualitative case study of emergency access control override mechanisms from hospital settings is a subject for future work. Another potential direction for future research could include log auditing to determine if by using real-time resource access log analysis and enforcement of logging obligations, we can limit the extent of uncertainty of the system state following an emergency access session, and allow for recovery to a known safe and secure state. Therefore, implementation and performance testing of such a system is included in future work plans as well.

[6] Recall the policy `polSetBtgFlag` in 5 returns deny only if the BTG flag is set to True. In all other cases it returns permit regardless of what group the resource *btg* belongs to or what the role of requesting user is. Similarly, the policy `polModifyAuthPolicy`, which defines access control policy for authorization policies, returns permit only if the requesting user is a system administrator and the system is normal state. Otherwise, it returns deny.

References

[1] King, A.L., Procter, S., Andresen, D., Hatcliff, J., Warren, S., Spees, W., Jetley, R.P. *et al.* (2009) An open test bed for medical device integration and coordination. In *International Conference on Software Engineering (ICSE) Companion*.

[2] Tasali, Q., Chowdhury, C. and Vasserman, E.Y. (2017) A flexible authorization architecture for systems of interoperable medical devices. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*.

[3] King, A., Arney, D., Lee, I., Sokolsky, O., Hatcliff, J. and Procter, S. (2010) Prototyping closed loop physiologic control with the medical device coordination framework. In *ICSE Workshop on Software Engineering in Health Care (SEHC)*.

[4] National Electrical Manufacturers Association (2013), Manufacturer disclosure statement for medical device security (MDS2), HIMSS/NEMA Standard HN 1-2013.

[5] Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R. and Chandramouli, R. (2001) Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* **4**(3).

[6] Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R. and Scarfone, K. (2014), Guide to attribute based access control (ABAC) definition and considerations, NIST Special Publication 800-162.

[7] Hu, V.C., Kuhn, D.R. and Ferraiolo, D.F. (2015) Attribute-based access control. *IEEE Computer* **48**(2).

[8] (2008), Medical Devices and Medical Systems-Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE)-Part 1: General requirements and conceptual model, ASTM F2761.

[9] Hatcliff, J., King, A., Lee, I., MacDonald, A., Fernando, A., Robkin, M., Vasserman, E.Y. *et al.* (2012) Rationale and architecture principles for medical application platforms. In *International Conference on Cyber-Physical Systems (ICCPS)*.

[10] John Hatcliff, Eugene Y. Vasserman, T.C. and Whillock, R. (2018) Challenges of distributed risk management for medical application platforms. In *IEEE Symposium on Product Compliance Engineering (ISPCE)*.

[11] OASIS (2013), eXtensible Access Control Markup Language (XACML) version 3.0, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[12] (2013), 45 CFR 164.312 - Technical safeguards. URL https://www.law.cornell.edu/cfr/text/45/164.312.

[13] U.S. Department of Health and Human Services Office for Civil Rights (2013), HIPAA Administrative Simplification.

[14] Povey, D. (1999) Optimistic security: A new access control paradigm. In *New Security Paradigms Workshop (NSPW)*.

[15] Rissanen, E., Firozabadi, B.S. and Sergot, M. (2004) Towards a mechanism for discretionary overriding of access control. In *International Workshop on Security Protocols (SPW)*.

[16] Ferreira, A., Cruz-Correia, R., Antunes, L., Farinha, P., Oliveira-Palhares, E., Chadwick, D.W. and Costa-Pereira, A. (2006) How to break access control in a controlled manner. In *IEEE International Symposium on Computer-Based Medical Systems (CBMS)*.

[17] Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zao, G., Chilro, R. and Antunes, L. (2009) How to securely break into RBAC: The BTG-RBAC model. In *Annual Computer Security Applications Conference (ACSAC)*.

[18] Brucker, A.D. and Petritsch, H. (2009) Extending access control models with break-glass. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*.

[19] Nazerian, F., Motameni, H. and Nematzadeh, H. (2019) Emergency role-based access control (E-RBAC) and analysis of model specifications with Alloy. *Journal of information security and applications* **45**.

[20] Helal, M.R. (2017) *Efficient Isolation Enabled Role-Based Access Control for Database Systems*. Ph.D. thesis, University of Toledo.

[21] Axiomatics (2015), Axiomatics language for authorization (ALFA), https://www.axiomatics.com/solutions/products/authorization-for-applications/developer-tools-and-apis/192-axiomatics-language-for-authorization-alfa.html. (Accessed on 2/21/2017).

[22] Mankai, M. and Logrippo, L. (2005) Access control policies: Modeling and validation. In *NOTERE Conference (Nouvelles Technologies de la Répartition)*.

[23] Hwang, J., Xie, T., Hu, V. and Altunay, M. (2010) ACPT: A tool for modeling and verifying access control policies. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*.

[24] Fisler, K., Krishnamurthi, S., Meyerovich, L.A. and Tschantz, M.C. (2005) Verification and change-impact analysis of access-control policies. In *Proceedings of the International conference on Software engineering (ICSE)*.

[25] Ma, J., Zhang, D., Xu, G. and Yang, Y. (2010) Model checking based security policy verification and validation. In *International Workshop on Intelligent Systems and Applications*.

[26] Bravo, L., Cheney, J. and Fundulaki, I. (2008) Accon: checking consistency of xml write-access control policies. In *Proceedings of the International conference on Extending database technology: Advances in database technology*.

[27] Aqib, M. and Shaikh, R.A. (2015) Analysis and comparison of access control policies validation mechanisms. *International Journal of Computer Network and Information Security* **7**(1).

[28] Siriwardena, M. (2017), Balana, https://github.com/wso2/balana. (Accessed on 1/12/2017).

[29] Lei, Y., Kacker, R., Kuhn, D.R., Okun, V. and Lawrence, J. (2007) IPOG: A general strategy for t-way software testing. In *IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS)*.

[30] NIST (2018), Access control policy tool (ACPT), https://www.nist.gov/programs-projects/access-control-policy-tool-acpt. (Accessed on 3/13/2018).

[31] Cimatti, A., Clarke, E., Giunchiglia, F. and Roveri, M. (1999) NuSMV: A new symbolic model verifier. In *International conference on computer aided verification (CAV)*.

[32] NIST (2018), Automated combinatorial testing for software (ACTS), https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software.

[33] Martin, E. and Xie, T. (2007) A fault model and mutation testing of access control policies. In *International conference on World Wide Web (WWW)*.

[34] Xu, D., Shrestha, R. and Shen, N. (2018) Automated coverage-based testing of XACML policies. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*.