Project Report
Real-Time Fault Detection in Access Control Rules Using Logic Circuits

Authors: Yatish Dubasi, Qinghua Li, Jia Di, University of Arkansas

Date of Release: January 31, 2024

Contact: Dr. Qinghua Li, qinghual@uark.edu

Acknowledgment: This work was supported in part by the U.S. National Institute of Standards and Technology (NIST) under Award No. 70NANB20H164. Thank Dr. Vincent C. Hu for his insightful feedback to this report. Thanks also go to Bryce Mendenhall for working on the early stages of the project, and to SangYun Kim for gathering part of the evaluation results.

1 Introduction

Access Control (AC) is a widely used security measure to protect commercial and government information systems. It controls which subjects (e.g., processes and users) have access to which resources based on AC policies. A policy consists of a set of rules where each rule assigns authorized subjects a permission (i.e., permit or deny) to access objects under certain environment conditions of the protected system. Specifying correct AC policies is crucial for protecting the target information systems; however, it has become a highly challenging task for AC policy makers as the protected systems become more and more complex such that a policy usually contains a large number of rules and is easy to contain faults. This challenge is being aggravated by the trend of incorporating Big Data, Cloud, Internet of Things (IoT), Smart Grid, and other distributed systems into intelligent systems. Faults in AC policies not only can leak sensitive information to unauthorized parties, but also disable authorized access to information. For example, one rule permits an access request but another rule denies it. Such faults may have catastrophic consequences. Thus, faults in AC rules must be detected and fixed in a timely manner.

However, fault detection using traditional approaches (see [1] for a survey) such as Model Checkers, Multi-Terminal Binary Decision Diagrams (MTBDD), Formal Methods, Mutation Testing, Automated Combinatorial Testing, and Pseudo-exhaustive Testing is time-consuming for AC policies with a large number of rules, due to the exploration of a large rule space. More importantly, traditional approaches usually work after the entire policy is made. They can detect the existence of faults but usually cannot identify which rule caused the fault.

Extending upon the prior work [2], this project further studies and implements logic circuit based real-time detection schemes to detect AC rule faults, which is fundamentally different from traditional formal methods such as model checking. We refer to our implemented tool as LogicDetect.

LogicDetect has three major advantages: 1) due to the high computational efficiency of logic circuits especially when they are implemented in hardware, it enables a fundamental transition from the traditional after-the-effect detection to real-time detection, a strategy deemed too costly and impractical for traditional methods; 2) LogicDetect takes advantage of the parallelism and Boolean expressions in logic circuit simulations and is able to reduce the computation cost by orders of magnitude compared to traditional techniques; and 3) LogicDetect can be implemented in physical electronic devices embedded/incorporated into the protected systems such that the AC fault checking is local, independent of the availability of high-speed network or high performance computers. It can be used in industrial control systems, Internet-of-Things, military systems, as well as regular information technology systems.

The remainder of this report is organized as follows. Section 2 provides a high-level overview of the logic circuit-based AC fault detection approach. Section 3 describes the logic circuits for static AC policies, properties, and models, and discusses how faults are detected. Section 4 describes the logic

circuits for dynamic AC polices, properties, and models, and discusses how faults are detected. Section 5 describes our implementation. Section 6 presents evaluation results. Section 7 concludes the report.

2 Solution Overview

The basic idea is to construct logic circuits (LC) to simulate/emulate AC rules. Intuitively, AC rules can be expressed by Boolean expressions that operate on AC rule variables. When specifying rules in a LC, each LC gate is a logic operation connecting AC variables enforced by the rules in a Boolean expression. Each AC rule in a LC generates a permission output. A positive output after changing the input variables of the LC from 0 to 1 represents a "grant" permission; a negative output after changing the input variables of the LC from 0 to 1 represents a "deny" permission. For example, the LC in Fig. 1 shows three simple AC rules: subject $s1$ is granted to perform action a on object $o1$ and $o2$, and subject $s2$ is granted to perform action a on object $o1$. The Boolean equivalences of the three rules are: $s1$ & a & $o2 = p1$, $s1$ & a & $o1 = p2$, $s2$ & a & $o1 = p3$. Triggering a, $s1$, $s2$, $o1$, and $o2$ of the LC will generate positive permission outputs on $p1$, $p2$, and $p3$ of the three AC rules. Triggering $s2$, a, and $o2$ will not activate any positive permission output since there is no $s2$ & a & $o2$ rule.

While adding AC rules to the policy, the corresponding circuits are constructed. When a new rule is added, the input switches representing this rule will be triggered on the LC and faults exist if the fault-checking output is positive. Specifically, two separate LCs are needed for fault detection: Grant LC and Deny LC, each of which implements all rules with grant and deny permissions respectively, as shown in Fig. 2. To detect if a newly added rule generates a permission conflict, the subject s, action a, and object o of this rule are triggered on both Grant LC and Deny LC, wherever applicable. If both LCs output a positive permission, which means both grant and deny permissions for the access represented by $\{s, a, o\}$ are present in the policy, a collision fault is detected.

The above process can be used to detect most of the faults. For missing privilege faults and cyclic inheritance faults, special logic circuit design is needed, which will be described in this report too.

In fault detection, when we trigger the inputs for a rule, we must also trigger the chain of privilege inheritors from the subject(s) as well as any attributes that are linked to those subjects and inheritors. This is needed in order to detect collision faults caused by privilege inheritance or attribute-based rules [3]. Attribute links are relationships between attributes/subjects. An example of an attribute link is the statement that John is a teacher, where John is a subject and teacher is an attribute. Similarly, when adding AC properties such as privilege inheritances or attribute links, we must apply them to the existing logic circuit to whichever rules they apply to, and we must re-check the existing policy for faults again as these new properties could cause faults.

Our approach can also be used to enforce AC rules, i.e., checking whether an access request should be permitted or denied. To do it, the inputs of the access request will be triggered on our circuit, and then we can observe the output of the Grant LC. If the output is high, the request is permitted; else, the request is denied.

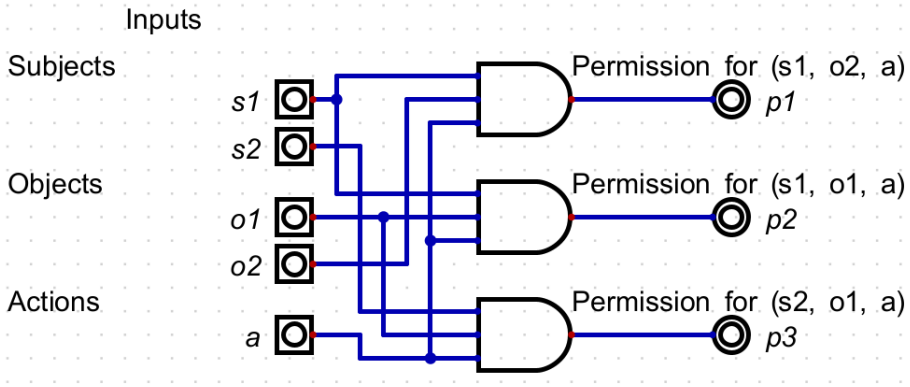

Figure 1: AC Rules Representations

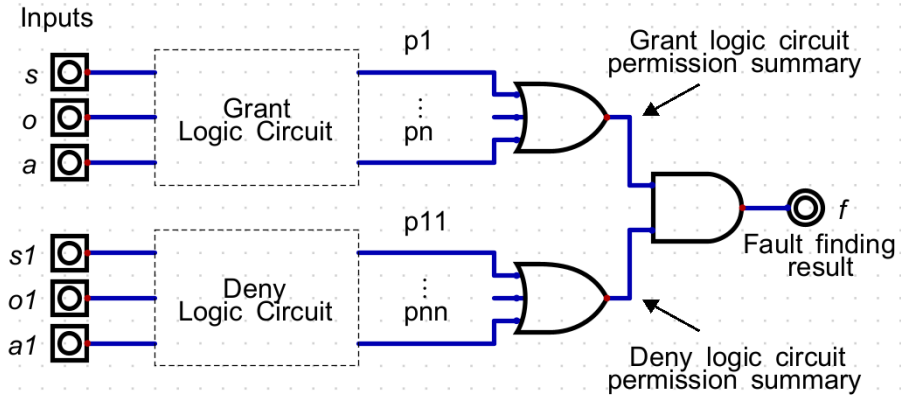

Figure 2: Permission Collision Fault

3 Logic Circuit Design and Fault Detection for Static AC Polices and Models

This section describes how to simulate static AC rules and properties with logic circuits, and how to detect faults with the circuits.

3.1 Simple AC Rules

Simple rules can be either grant or deny rules. They either grant or deny an access request, i.e., permission for a subject to perform an action on an object.

3.1.1 Potential Faults

A collision fault can occur if a rule states that a subject is granted access to perform an action on an object, but another rule states that the subject is denied access to perform that action on that object.

3.1.2 Logic Circuit Representation and Fault Detection

Simple AC rules consist of usually 3 parts: subject, object, action. These inputs will be AND-ed together to form the simple rule's logic circuit. For example, a rule states that John is allowed to read a document. The subject is "John"; the object is "document"; the action is "read". Therefore the rule's logic representation will be "John AND document AND read" (see Fig. 3). Since this rule is a Grant rule, the logic representation will be placed in the Grant LC. If the rule was a deny rule, such as statin that John was not allowed to perform a certain permission, it would be placed in the Deny LC.

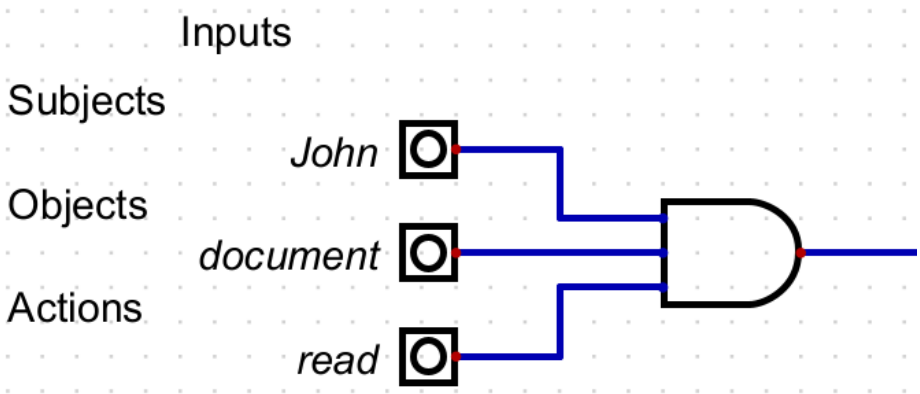

Figure 3: Simple Rule Logic Representation

After a rule is added to the LC, the corresponding inputs (i.e., subject, object, action) will be triggered. If a positive output is produced in both the Grant LC and the Deny LC, this new rule is causing a fault.

For example, suppose the following two rules are added to the policy (they are formatted as follows: subject, object, action):

- Grant: John, document, read
- Deny: John, document, read

These two rules will cause a collision as John is being granted and also denied access. This fault will be detected when the second rule is added to the policy, regardless of the order the rules were entered in, as there will be an output of high from the Grant LC and also an output of high from the Deny LC when triggering the inputs of the rule.

3.1.3 Rule Enforcement

Rule enforcement is checking whether an access request should be permitted or denied. When the inputs of an access request are triggered, we check to see if an output of high occurs in the Grant LC. If so, the request is permitted. For simple AC rules, we will utilize the fault detection logic representation for rule enforcement as well. For the remaining document, we will only include this section if the rule enforcement differs from the fault detection logic representation.

3.2 Attribute-based Access Control Model

Attribute-based model's rules are enforced on attributes instead of a traditional "subject" or "object". They are commonly present in Attribute-based Access Control (ABAC) models [3]. We can use attribute-based rules with any of the static and dynamic rules by simply replacing the subjects or objects with attributes. Examples of attributes are teacher, staff, "born before 1999", "US citizen", and so on. For attribute-based rules, the logic circuit design depends on the underlying type of AC policy or model; it is the same as those described in previous and following sections except to replace subjects or objects with attributes. The fault detection process is also similar. We do not repeat it here to avoid redundancy.

3.3 Multi-Level Security Model

Multi-level security (MLS) model [4] assigns each object a security level and gives each subject a security clearance level. The subjects then have access to certain objects based on their security levels.

3.3.1 Potential Faults

A fault can occur if a rule contradicts a MLS' access right by denying that permission. For example, if a security level was granted access to an object, but then later that security level is denied access to that object, this will be a fault.

3.3.2 Logic Circuit Representation and Fault Detection

We simply treat the security levels as attributes. For example, suppose there are three security levels in order of increasing importance: confidential, secret, and top secret. Suppose there are three objects: confidential object (CO), secret object (SO), and top secret object (TSO). Now suppose we are implementing the Bell-LaPadula model, where subjects cannot read an object that requires a

higher security level (no read up) and cannot write to an object that requires a lower security level (no write down). MLS can now be represented the same way attribute-based rules; the logic representation for the read rules will be as follows: "top secret AND TSO AND read", "(top secret OR secret) AND SO AND read", "(top secret OR secret OR confidential) AND CO AND read". See Fig. 4.

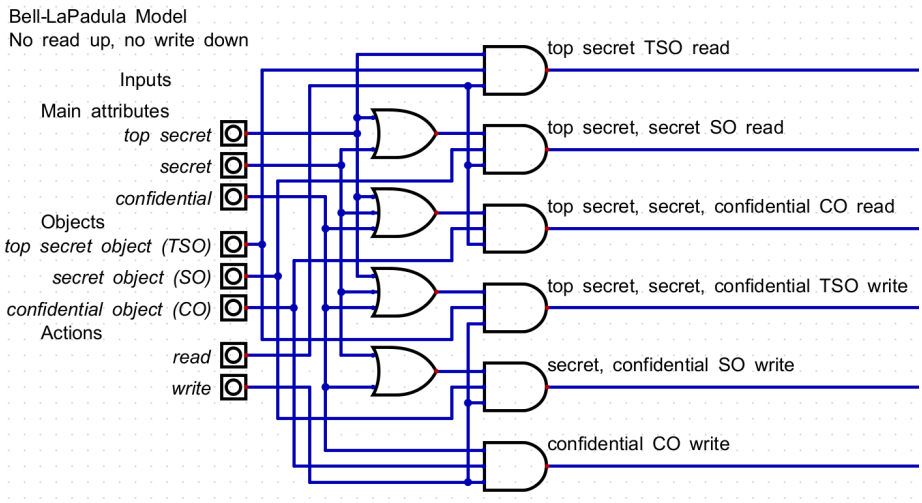

Figure 4: Multi-Level Security Logic Representation

To illustrate a collision fault with an MLS rule, suppose we have the following two rules, where the first is the MLS rule in our previous example (Bell-LaPadula model) and the second is a simple rule.

- Grant: MLS Bell-LaPadula model
- Deny: top secret, TSO, read

This fault will be detected during the fault detection procedure as there will be an output of high from the Grant LC and also an output of high from the Deny LC when triggering the inputs of one of the rules.

3.4 Inheritance

Inheritance is an access control property that automatically applies any rule from the tribute/parent subject to another beneficiary/child subject. For example, if manager inherits from employee, any rule that is applied to employee will also then be applied to manager.

3.4.1 Potential Faults

Suppose there is an inheritance relationship in the AC policy. A fault can then occur when the tribute or beneficiary is granted and denied a certain permission. We describe how to detect such a fault in Section 3.4.2. A cyclic inheritance fault can also occur due to inheritance relationships. A cyclic inheritance fault occurs when the chain of inheritance from a subject goes all the way around and comes back to the original subject. In order to detect cyclic inheritance faults, we need to create a separate logic circuit. Details regarding this circuit will be covered in Section 3.4.3.

3.4.2 Logic Circuit Representation and Fault Detection

We can represent inheritance relationships in our logic circuits as well. When we process a rule, we will check if the subject has any inheritors; if the subject does have inheritors, we will set an OR condition between the main subject and their inheritors as the entry to the rule's logic representation. Additionally, if the inheritors of the subject themselves also have inheritors, we will process those as well until all the inheritors are processed. For example, suppose that manager inherits from employee, director inherits from manager, and owner and editor inherit from director. Now suppose our rule stated that employee is allowed to read a folder. Our rule's logic representation would be "(employee OR manager OR director OR owner OR editor) AND folder AND read" (see Fig. 5). Since this rule is a Grant rule, the logic representation will be placed in the Grant LC. If the rule was a deny rule, it would be placed in the Deny LC.

Inheritance can cause faults. Here, we illustrate a collision fault. Suppose manager inherits from employee and a policy has the following rules:

- Grant: employee, folder, read
- Deny: manager, folder, read

Due to the inheritance, manager should be able to read the folder too so the Grant LC will output high (see Fig. 5). However, the deny rule will cause the Deny LC to generate a high output. Then the fault will be detected.

3.4.3 Cyclic Inheritance

Cyclic inheritance faults occur when the chain of inheritance from a subject wraps back around to the original subject. For example, suppose manager inherits from employee, director inherits from manager, and employee inherits from director. The inheritance chain from employee goes to manager, then to director, then it wraps back around to employee; this causes a cyclic inheritance fault. In order to detect this fault, we will create a separate logic circuit that simulates only inheritance relationships. The inputs in this logic circuit will be only the subjects/attributes involved in inheritances. This logic circuit should be designed in such a way that, if one input it triggered, it should activate the entire chain of inheritances, and if the chain wraps back around to the original

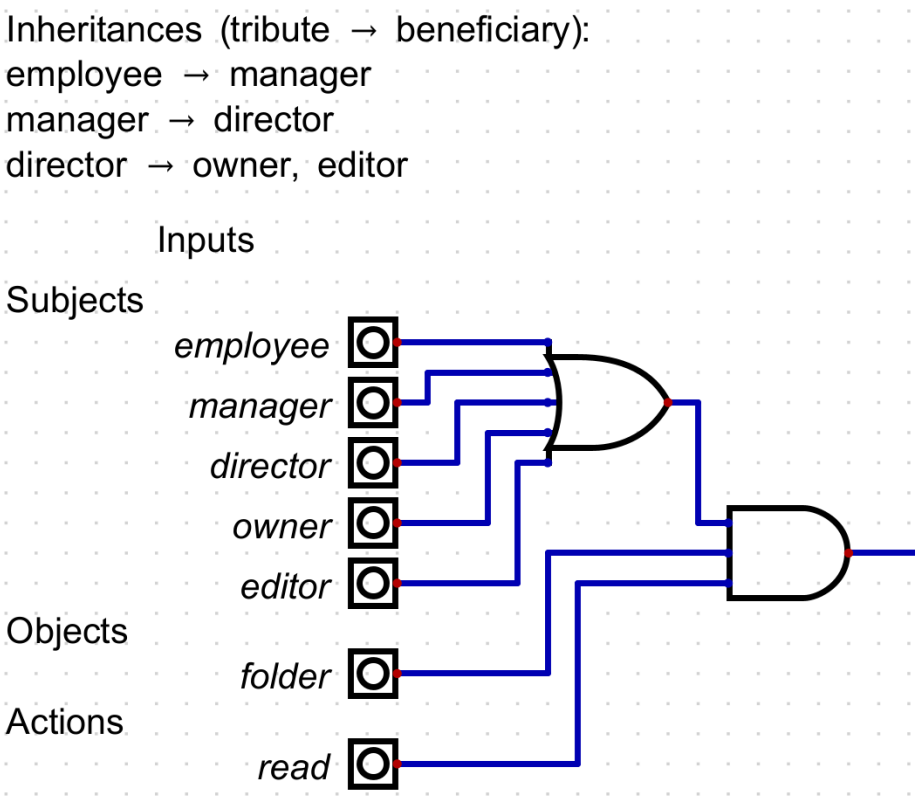

Figure 5: Rules with Inheritance Logic Representation

input we will detect that there is a cyclic inheritance fault. The logic representation of each inheritance relationship is as follows: "subject OR {output value of any subject it directly inherits from}"; where the "output value" of a subject will also be the output of such a logic representation. If a subject is a beneficiary, the subject will need this logic representation. If a subject is only a tribute and not a beneficiary, we will use the input itself directly as the "output value" for that subject. We then check for cyclic inheritance with the following logic: "subject AND {output value of any subject it directly inherits from}". If the output of that is high, we detect a cyclic inheritance fault. We can see an example of these circuits with and without a fault in Fig. 6. In Fig. 6, the "Gounded Output" is simply used to compile all the regular output values of subjects; the value of this output does not matter for fault detection.

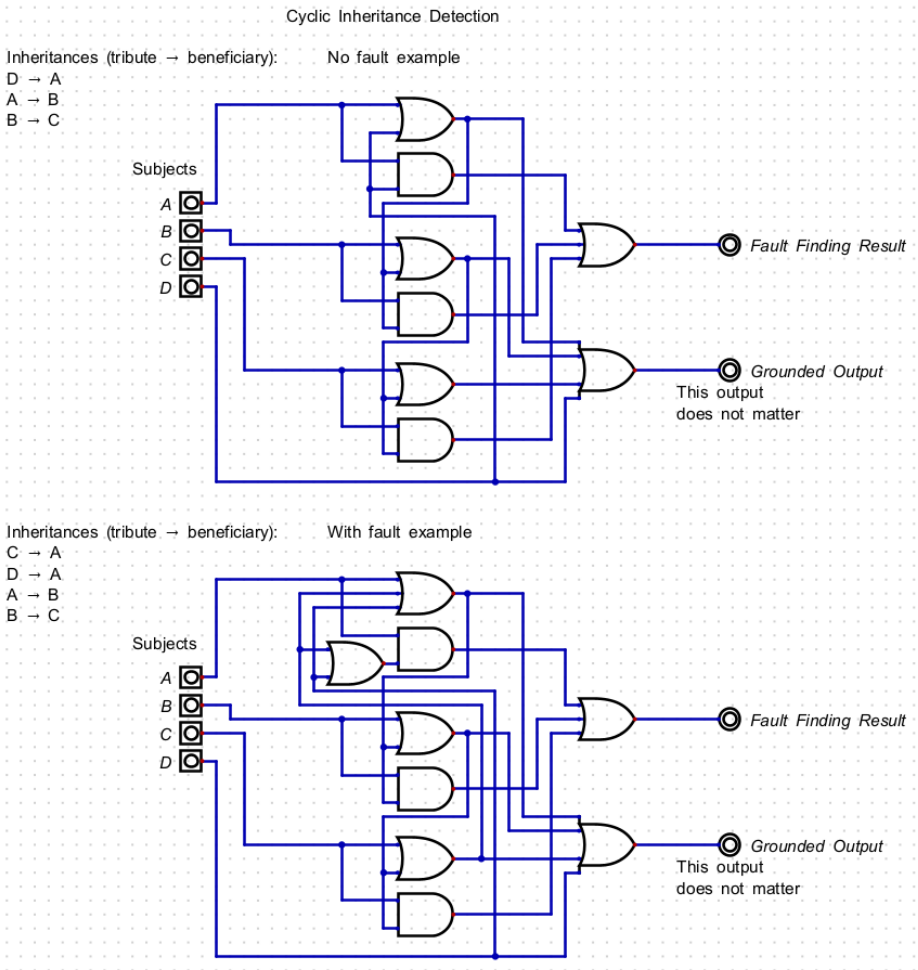

Figure 6: Cyclic Inheritance Detection

4 Logic Circuit Design and Fault Detection for Dynamic AC Polices and Models

This section describes how to simulate dynamic AC rules and properties with logic circuits, and how to detect faults with the circuits.

4.1 n-person-control Model

In an n-person-control (nPC) model [5], n subjects, from a group of s or more subjects, must be active in order to access a certain object. For example, a 2-person-control rule states that any two subjects from the group of employee,

manager, and director together are allowed to read a folder; however, the subjects when alone cannot read the folder. The subject count, n, will be initialized and fixed the first time an n-person control rule is created.

4.1.1 Potential Faults

Suppose there is an n-person-control rule in the AC policy. A fault can then occur when a proper subset of the s subjects is granted access for the same object and action. In the example above, if another rule says employee alone is allowed to read the folder, it will induce a fault.

4.1.2 Logic Circuit Representation and Fault Detection

For ease of presentation, let us consider the example rule that any two subjects from the group of employee, manager, and director together are allowed to read a folder. As shown in Fig. 7, we will add a deny logic representation to the Deny LC for each subject in the group. In our example, we will add the following logic representations to the Deny LC "(employee OR manager OR director) AND folder AND read".

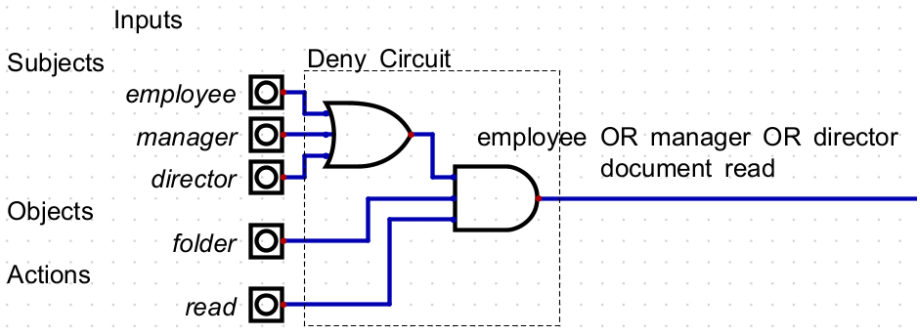

Figure 7: n-person-control Logic Representation

For example, suppose the following two rules are added to the policy where the first rule is a 2-person-control rule:

- Grant 2PC: (employee, manager, director), folder, read
- Grant: manager, folder, read

Suppose the 2-person-control rule is added first. Then, when adding the second rule, a fault will be detected, because it will conflict with the "(employee OR manager OR director) AND folder AND read" circuit added to the Deny LC when the first rule was placed in the policy.

4.1.3 Rule Enforcement

For rule enforcement, logic representations will only be added to the Grant LC; we will add a grant logic representations for each n subject subset. The example rule's logic representation will be "((employee AND manager AND NOT director) OR (employee AND NOT manager AND director) OR (NOT employee AND manager AND director)) AND folder AND read". See Fig. 8.

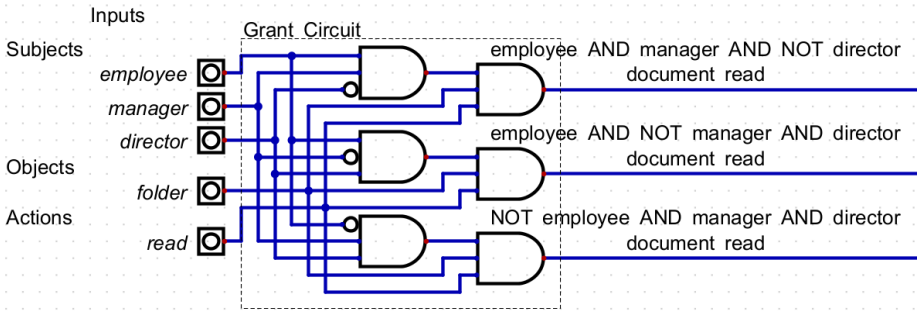

Figure 8: n-person-control Logic Representation

4.2 Mutual Exclusion Model

In a mutual exclusion model [5], a subject from a group of s subjects is allowed to access a certain object at a time, but multiple of them cannot access the object at the same time. For example, employee and manager are not allowed to read a folder at the same time, but either of them alone is allowed to read it. Additionally, the first subject that performs this rule will claim the permission exclusively for themselves. So in our previous example, if manager reads the folder first, only manager will have access to the folder. Mutual exclusion is a dynamic rule. Rules that enable different access rights based on dynamic situations are called dynamic rules. Here, we define access rights as potential permissions that a subject has. For example, if a rule allows employee or manager to read a folder, we will have the following access rights: (employee, folder, read), (manager, folder, read). Mutual exclusion rules are dynamic rules since the access rights of each subject change over time and depend on the system states.

4.2.1 Potential Faults

A fault can occur if another AC rule denies a subject from the mutual exclusion group to access that object. This is a fault because the subject alone should be allowed to access the object. Another kind of fault can occur if another rule in the policy grants multiple subjects in that mutually exclusive group to access the object at the same time.

4.2.2 Logic Circuit Representation and Fault Detection

For example, suppose a rule states that only one person out of employee or manager is allowed to read a folder. When performing fault detection, we will use an OR between the subjects. The logic representation when performing fault detection would be "(employee OR manager) AND folder AND read", and it will be added to the Grant LC. See Fig. 9. Next, we show how fault is detected.

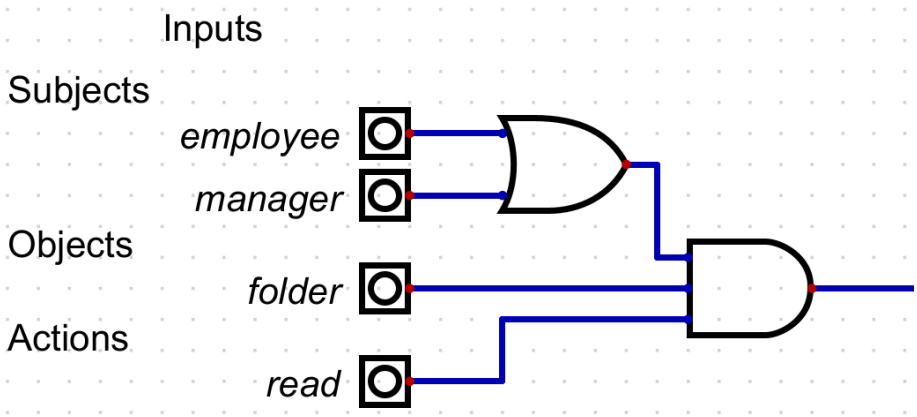

Figure 9: Mutual Exclusion Logic Representation

If we want to induce a collision fault with a mutual exclusion rule, suppose we have the following two rules, where the first is a mutual exclusion rule and the second is a simple rule.

- Grant: employee XOR manager, folder, read
- Deny: manager, folder, read

This fault will be detected during the fault detection procedure as there will be an output of high from the Grant LC and also an output of high from the Deny LC when triggering the inputs of one of the rules.

For the second type of fault discussed in Section 4.2.1, suppose one rule states that director, manager, and employee are in one mutually exclusive group for reading a folder. That means the circuit (director OR manager OR employee) AND folder AND read is added to the Grant LC. If another rule states that director and manager together are allowed to read the folder, then this rule is equivalent to a 2-person-control rule. Similar to the example in Fig. 7, circuit (director OR manager) AND folder AND read is added to the Deny LC. These two circuits will cause a collision. Hence, the fault will be detected.

4.2.3 Rule Enforcement

For rule enforcement, the logic representation would be "{subject} AND {object} AND {action} AND NOT {output of other access rights' flip flops}". In this example, the rule's logic representation will be "employee AND folder AND read AND NOT {output of flip flop for manager can read folder}". For simulation of dynamic rules, we use flip flops in our logic circuits to maintain the dynamic states. See Fig. 10.

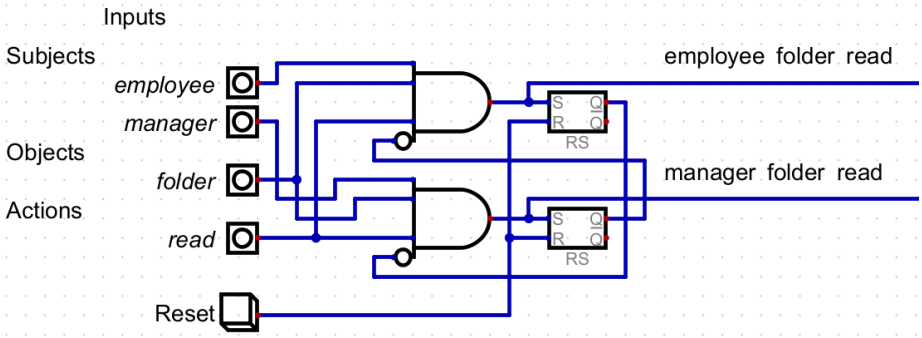

Figure 10: Mutual Exclusion Logic Representation

4.3 Separation of Duty Model

In Separation of Duty (SoD) model [6], rules have multiple subjects, one object, and multiple actions. All the subjects are allowed to perform any of the actions on the object; however, each subject is only allowed to choose one action to be able to perform. Once one subject performs an action, that action will be exclusive to that subject, and other subjects must choose from the remaining actions. SoD rules are dynamic rules since the access rights of each subject change over time and depend on the system states.

4.3.1 Potential Faults

Despite separation of duty, the subjects are still allowed to perform the actions on the object. Thus, a fault can occur when another rule denies any of the subjects from taking any of the actions over the object.

4.3.2 Logic Circuit Representation and Fault Detection

For simulation of dynamic rules, we can use flip flops in our logic circuits to maintain the dynamic states. For example, suppose a rule states that employee or manager are allowed to read or write a folder, but there must be a separation of duty between employee and manager. We would have four access rights in this rule: "employee can read folder", "employee can write folder", "manager

can read folder", and "manager can write folder". The logic representation of each access right will be followed by a flip flop to denote an action being claimed by that subject. The logic representation of each access right would be "{subject} AND {object} AND {action} AND NOT {output of other access rights' flip flops containing the same subject} AND NOT {output of other access rights' flip flops containing the same action}". If we look at the access right of "employee can read folder" in our example, the logic representation will be "employee AND folder AND read AND NOT {output of flip flop for employee can write folder} AND NOT {output of flip flop for manager can read folder}". The logic representation will be placed in the Grant LC. See Fig. 11. During fault detection, we need to reset these flip flops back to the reset state before triggering each individual rules' inputs. This will make sure that all actions could be claimed by any subjects at any time, allowing for proper fault detection.

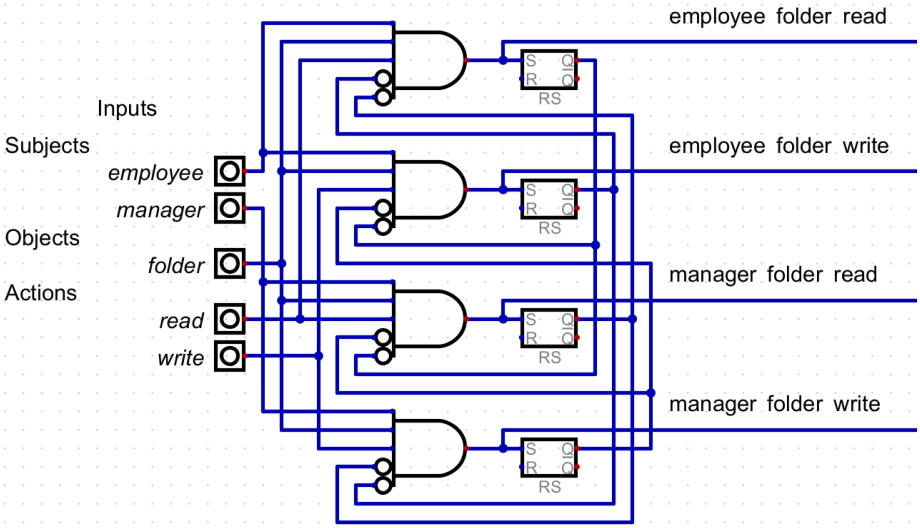

Figure 11: Separation of Duty Logic Representation

In order to illustrate collision fault detection with a SoD rule, suppose we have the following rules:

- Grant: SoD (employee, manager), folder, (read, write)
- Deny: manager, folder, read

employee and manager could claim the actions of read or write on the folder. The second Deny rule would conflict with manager's ability to claim the read action on folder. This fault will be detected during the fault detection procedure as there will be an output of high from both the Grant LC and the Deny LC when triggering the inputs of one of the rules.

4.4 Object-based Separation of Duty Model

Object-based separation of duty model's rules are similar to regular SoD rules. The difference is that object-based SoD rules will have multiple subjects, multiple objects, and one action; subjects will claim an object instead of an action as in regular SoD.

4.4.1 Potential Faults

Suppose there is an object-based SoD rule in the AC policy. A fault can occur when an access right from the object-based SoD rule is later on added as a deny rule.

4.4.2 Logic Circuit Representation and Fault Detection

Similar to regular SoD, the logic representation of each access right in an object-based SoD rule would be "{subject} AND {object} AND {action} AND NOT {output of other access rights' flip flops containing the same subject} AND NOT {output of other access rights' flip flops containing the same object}". The logic representation will be placed in the Grant LC. See Fig. 12.

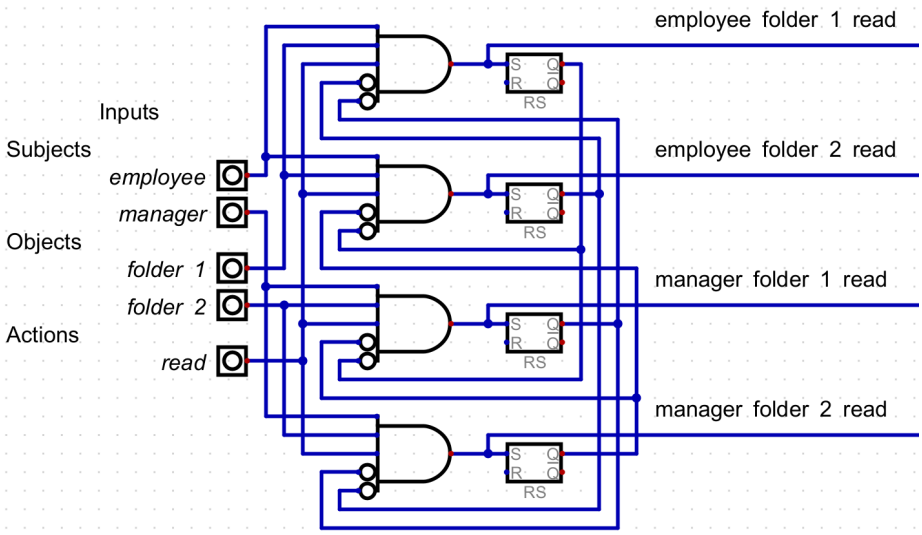

Figure 12: Object-based Separation of Duty Logic Representation

The fault detection for this rule is very similar to regular SoD. To illustrate a collision fault with object-based SoD, suppose we have the following rules.

- Grant: object-based SoD (employee, manager), (folder 1, folder 2), read
- Deny: manager, folder 1, read

This fault will be detected during the fault detection procedure as there will be an output of high from both the Grant LC and the Deny LC when triggering the inputs of one of the rules.

4.5 Workflow Model

Workflow model's rules are also dynamic rules [6]. A workflow rule consists of multiple simple rules that must be done in order. For example, a teacher can create an exam, then and only then a student can take the exam, then and only then a grader can grade the exam.

4.5.1 Potential Faults

Suppose there is a workflow rule in the AC policy. A fault can occur when an access right in the workflow rule is denied by another rule.

4.5.2 Logic Circuit Representation and Fault Detection

Let us consider the above example. As illustrated in Fig. 13, the logic representation of each access right will be followed by a flip flop to denote that the access right was granted. The logic representation of each access right in a workflow rule would be "{subject} AND {object} AND {action} AND {output of directly previous access right's flip flop if present}". For the above example, the logical representations will be the following: "teacher AND exam AND create", then "student AND exam AND take AND {output of flip flop for teacher can create exam}", then "grader AND exam AND grade AND {output of flip flop for student can take exam}". The logic representation will be placed in the Grant LC. During fault detection, we always make sure these flip flops are in their latched output state (opposite of reset state). This will make sure that all of the sub-rules in the workflow could be triggered during fault detection.

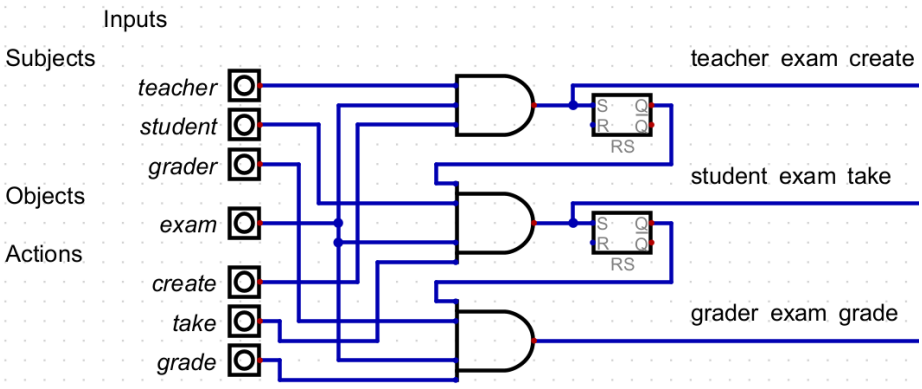

Figure 13: Workflow Logic Representation

To illustrate a collision fault with the workflow model, suppose we have the following rules:

- Grant: Workflow (employee, folder, create) → (manager, folder, read)
- Deny: manager, folder, read

The deny rule conflicts with manager's ability to complete the workflow and read the folder. This fault will be detected during the fault detection procedure as there will be an output of high from the Grant LC and the Deny LC when triggering the inputs of one of the rules.

4.6 Conflict of Interest Model

In Conflict of Interest (COI) model, the goals or utility of multiple objects are incompatible. For example, if an employee is working for multiple competing companies and has access to their proprietary information, that employee could use information from one company and use it at another to potentially cause a loss for the first company. In access control, setting COIs between objects allow the access control model to prevent a subject from having access to two or more objects that have COIs [5].

4.6.1 Potential Faults

For COI, no faults can occur during authorization time directly related to COI. Regular fault detection can occur without regard to any restrictions imposed by COI, as COI only takes effect during run time.

4.6.2 Logic Circuit Representation and Fault Detection

Since no faults can occur due to COI during authorization time, we will not include any logic representations of COI when performing fault detection.

4.6.3 Rule Enforcement

When we process a grant rule into the policy, we will check if the object that is being granted has any COIs; if the object does have COIs and the subject already has grant rules associated with those objects, we will add flip flops after each access right containing objects with conflicts; then we can have the following logic representation for each of those access rights: "{subject} AND {object} AND {action} AND NOT {output of other access rights' flip flops containing a conflicting object}". For example, suppose there is a COI between Apple secret files and Samsung secret files, and we have the following two rules.

- Grant: employee, Apple secret files, read
- Grant: employee, Samsung secret files, read

We will add the following logic representations to the Grant LC: “employee AND Apple secret files AND read AND NOT {output of flip flop for employee can read Samsung secret files}”, “employee AND Samsung secret files AND read AND NOT {output of flip flop for employee can read Apple secret files}”. See Fig. 14.

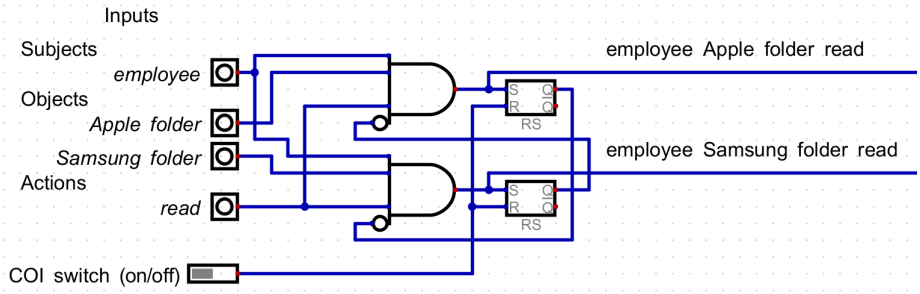

Figure 14: Conflict of Interest Logic Representation

5 Implementation

These logic circuits can be implemented on physical hardware, such as FPGA boards. We used the Xilinx Virtex-7 FPGA VC707 Evaluation Board for implementing on physical hardware. For ease of use and evaluations, we have also developed a software tool that simulates the behavior of these logic circuits and our fault detection methodology. This tool is titled ACLST (Access Control Logic Simulation Tool). ACLST is a command line interface tool that is implemented with Python 3. We can see the menu of ACLST in Fig. 15.

In this tool, a user can simulate an AC policy by adding or deleting rules and access control properties. This tool supports all of the access control rule types and properties described above.

Whenever a new rule or property is added to the policy, the tool will automatically check if the new rule will create any faults. If the new addition does not cause any faults, the rule will be added to the policy. If the new addition does induce a fault, ACLST will detect the fault and abort the addition; if a user needed the new rule, the user could instead delete the existing rules and/or properties that conflict with the new rule in order to insert the new rule. The tool is capable of detecting all of the faults described above.

In addition to fault detection, ACLST can also be used in order to simulate the enforcement of rules. When provided an access request, the tool can determine if the request should be granted or denied according to the AC policy. This is done by simulating the grant circuit and seeing if a request’s inputs trigger a grant output as high. The tool is able to appropriately simulate and maintain the states for dynamic rules as well.

Moreover, ACLST can display a diagram of what the logic circuit is for the current policy rules. This function was implemented based on SchemDraw [7]. Users can save this diagram as an image or file. A user could reference this diagram in order to construct the circuit on actual hardware if desired. An example of a ruleset and the generated diagram is shown in Fig. 16.

Lastly, policies can be exported or saved as files. These files can be loaded with ACLST in order to recover the policy and pick up where the user left of previously.

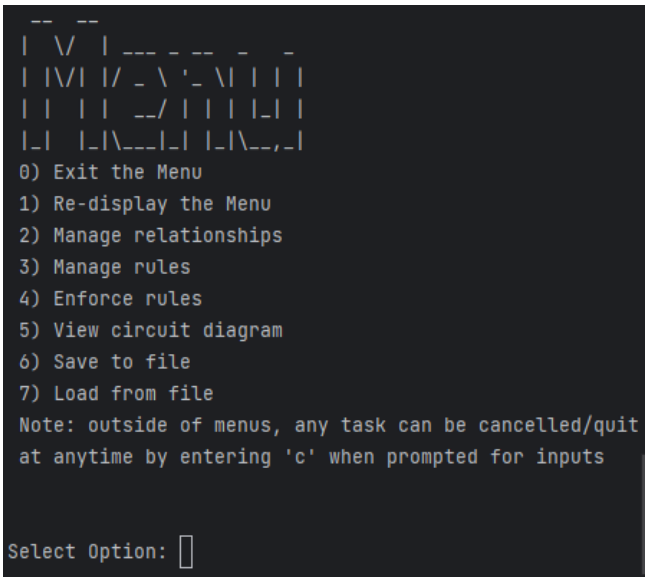

Figure 15: ACLST Command Line Interface

6 Evaluations

This section presents evaluations of our approach in performance and cost.

6.1 Complexity Analysis

First, we analyze the temporal and spatial complexity of simulating each AC rule or property as a logic circuit. Temporal complexity refers to how quick we can get the final result from our inputs, which is measured by the longest input-to-output path in the logic circuit. Spatial complexity refers to how large the system will be, which can be measured by the number of logic gates used in the logic circuit. Then, we analyze the complexity of processing a mix of m AC rules.

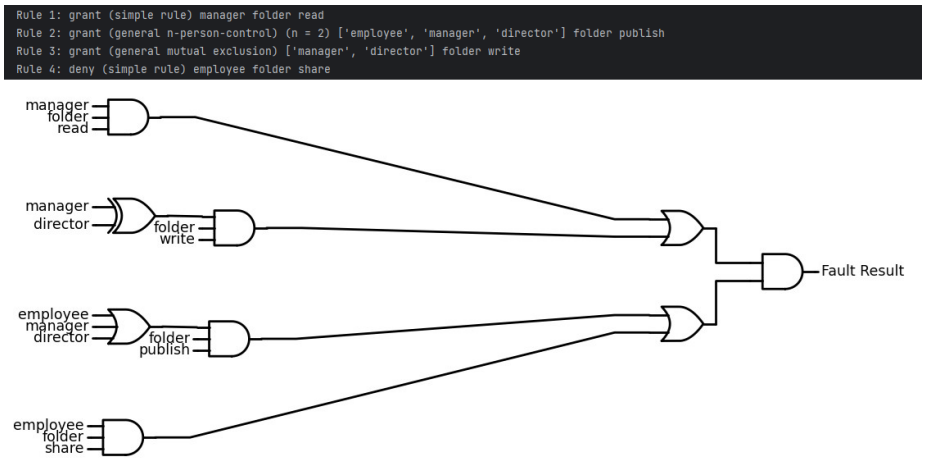

Figure 16: ACLST Example Ruleset and Generated Diagram

6.1.1 Complexity of Each AC Rule/Property

Simple Rules The temporal complexity of simple rules is O(1). This is because there is only one AND gate in the logic representation of simple rules. Similarly, the spatial complexity is also O(1) for a simple rule.

Attribute-based Rules Attribute-based rules have the same complexities as any subject-based rule or property, because we can simply replace the subject with an attribute in any rule type or property. Both the temporal and spatial complexities are O(1).

Multi-Level Security MLS models are represented with attribute-based rules. Thus, both the temporal and spatial complexities are O(1).

Inheritance If a subject or attribute that was given a rule had inheritors, the logic representation would increase by one gate per subject/attribute that has inheritors in the rule. However, the final temporal and spatial complexities will stay the same.

Next, we analyze the complexity of the special cyclic inheritance detection circuit. The temporal complexity is O(1) for an inheritance relationship. This complexity comes from an OR gate that checks if any of the direct inheritors of a subject are high; the next is an AND gate that checks if the output of the previous OR gate and the input subject value are both high in order to detect cyclic inheritance. The spatial complexity of this entire circuit is O(s) where s is the number of unique subjects that are involved in inheritance relationships. If the number of unique subjects involved in inheritance relationships does not change, the spatial complexity will not be affected by increases or decreases in the number of inheritance relationships. The spatial complexity is more precisely bound by $3{\cdot}s$. Here the number 3 comes from the previously mentioned two gates and another OR gate that determines the "output value" of a subject involved in inheritance relationships. The spatial complexity will not be

affected by increases in the number of relationships, because the relationships are mapped with wires to existing gates instead of using new gates.

n-person-control The temporal complexity of n-person-control rules is O(1), as there are two gates in the automatically added deny logic representation as part of n-person-control rules. The spatial complexity is also O(1) for an n-person-control rule.

Mutual Exclusion First, let us assume that we use RS flip flops as our flip flops in the logic representations. RS flip flops are composed of two NOR gates stacked on top of each other. The temporal complexity is O(1) regardless of the size of the mutual exclusion rule. The spatial complexity is O(s) where s is the number of subjects in the mutual exclusion rule. This is because a logic representation is made for each access right in the mutual exclusion rule.

Separation of Duty The temporal complexity will be O(1) regardless of the size of the SoD rule. The temporal complexity is formed by looking at the initial AND gate for an access right and the following NOR gate from the RS flip flop. The spatial complexity is O($s{\cdot}a$) where s and a are the number of subjects and the number of actions in the SoD rule respectively. The spatial complexity is more precisely bound by $3{\cdot}s{\cdot}a$, where the 3 comes from the AND gate of an access right and the two NOR gates in an RS flip flop, and the $s{\cdot}a$ denotes the number of access rights that will be generated from the SoD rule.

Object-based Separation of Duty The complexities of object-based SoD are similar to regular SoD. The temporal complexity is O(1). The spatial complexity is O($s{\cdot}o$) where s and o are the number of subjects and objects in the object-based SoD rule respectively.

Workflow The temporal complexity for workflow rules will also be O(1) regardless of the size the workflow rule. The temporal complexity is formed by looking at the initial AND gate for an access right and the following NOR gate from the RS flip flop. The spatial complexity would be O(a) where a is the number of access rights in the workflow rule. The spatial complexity is more precisely bound by $3{\cdot}a - 2$; the 3 comes from the AND gate of an access right and the two NOR gates in an RS flip flop. We subtract two at the end because the very last access right in the workflow rule will not need an RS flip flop to store its state.

Conflict of Interest The temporal complexity of rules containing objects with COIs will remain the same as the original rule, as the addition of a flip flop and condition for granting the permission will not increase the temporal complexity. Similarly, COI will not increase the spatial complexity as well.

6.1.2 Complexity for a Mix of m Rules

Next, we consider an AC policy of m rules that could be a mix of the rule types discussed above, and analyze the spatial complexity of representing these rules as well as the temporal complexity of detecting faults for the m rules. The analysis applies to any of the faults excluding cyclic inheritance. The detection of cyclic inheritance utilizes a different circuit, and its analysis was done in the previous section. Some complexities will use 'r' instead of 'm', where 'r' is the

number of access rights in the policy. For the majority of policies, r will be the same as m. However for SoD, workflow, and MLS rules, r can be larger than m. In SoD rules, there will be $s{\cdot}o$ number of access rights, where s is the number of subjects and o is the number of actions/objects. o is the number of actions for regular SoD rules, and o is the number of objects for object-based SoD rules. In workflow and MLS rules, the number of sub-rules will be the number of access rights.

Spatial Complexity In the worst case, the spatial complexity for an entire policy with mixed rules will be $O(r)$, where 'r' is the number of access rights in the policy.

Temporal Complexity If this fault detection system is run on physical hardware, when adding a rule, the runtime for checking for faults has constant complexity and bounded by the temporal complexity of the rule with the largest temporal complexity in the policy. Since the temporal complexity of checking each rule for all rule types is O(1), the the complexity for checking the faults induced by one rule is O(1). For the entire policy with 'r' number of access rights, the complexity is $O(r)$.

In software, ACLST simulates the behaviour of this system, and it has to loop through all of the existing rules in a policy while checking faults induced by each newly added rule. Therefore, ACLST's runtime for checking each newly added rule will have a linear complexity bounded by the number of rules already in the policy. For a policy with m rules, the complexity of checking for faults in the entire policy will be $O(m{\cdot}(m{+}1)/2)$, i.e., $O(m^2)$.

6.2 Experimental Evaluations

We have designed multiple comprehensive rulesets containing a diverse set of rules, rule types, and AC properties. These rulesets are based on various real-life domains such as software development and asset management systems. We also designed smaller rulesets based on academic, healthcare, and military systems. The rulesets can be seen in Section A, in the Appendix at the end of the report.

In order to test our solution's fault detection capabilities, we injected specifically curated fault-inducing rules/properties into our rulesets, by modify existing rules or properties in order to create various faults.

6.2.1 Evaluation of Correctness

We first tested whether our approach can correctly detect faults based on the ACLST tool. For the aforementioned rulesets and specially curated fault-inducing rules and properties, we found that ACLST was able to correctly detect all of the faults, which cover all the previously described fault types, AC policies/models, and AC properties.

6.2.2 Runtime Measurements

In order to evaluate the runtime, we created AC policies with various amounts of rules and took note of the runtimes when implemented on hardware and for ACLST to check for faults in all of the rules. We compared these runtimes to the runtimes of another access control policy fault detection tool called Access Control Policy Tool (ACPT) [8]. We took note of the runtimes for ACPT to check for faults in the same policies. These tests on hardware were run on the Xilinx Virtex-7 FPGA VC707 Evaluation Board. These tests on software were run on a computer with an AMD Ryzen 5 3600 6-Core Processor running at 3.6 GHz using 16 gb of ram, running Windows 11 Pro version 21H2.

Number of Rules	Runtimes for checking the entire policy (seconds) LogicDetect Hardware	ACLST	ACPT
1	2.534e-9	2.130e-5	1.019
10	2.461e-8	2.361e-4	1.021
20	4.887e-8	6.858e-4	1.017
30	7.325e-8	1.120e-3	1.007
40	9.738e-8	1.640e-3	1.010
50	1.215e-7	2.072e-3	1.018

Table 1: Runtimes: ACLST vs ACPT

We can see the results in Table. 1. We observed that, for the test cases we ran, ACPT takes around one second to do fault checking. However, the hardware implementation and ACLST seem to have much shorter runtimes when the number of rules in the policy are not extremely large.

7 Conclusion

We explored the extended capacities of simulating AC rules on a logic circuit by applying it to complex AC models. We discussed how to simulate various AC rule types and AC properties with logic circuits. We created an implementation that simulates the behavior of this fault detection logic circuit, ACLST. We evaluated our approach by checking if it properly detects faults against various AC policies with specially injected rules that should induce faults. Our tool was able to detect all of the tested faults against various AC rule types. We analyzed the temporal and spatial complexity of our fault detection scheme, and measured its running time. Lastly, we also evaluated ACLST's performance by comparing it with ACPT, a traditional model-based fault detection system.

References

[1] Vincent C Hu, Rick Kuhn, Dylan Yaga, et al. "Verification and test methods for access control policies/models". In: *NIST Special Publication* 800 (2017), p. 192.

[2] Vincent C Hu and Karen Scarfone. "Real-time access control rule fault detection using a simulated logic circuit". In: *2013 International Conference on Social Computing*. IEEE. 2013, pp. 494–501.

[3] Vincent Hu et al. "Guide to Attribute Based Access Control (ABAC) Definition and Considerations". In: *NIST Special Publication 800-162* (2019).

[4] *Committee on National Security Systems (CNSS) Glossary*. Committee on National Security Systems, 2015.

[5] Terry Mayfield et al. "Integrity in automated information systems". In: *National Security Agency, Tech. Rep* 79 (1991).

[6] Vincent C Hu, David Ferraiolo, D Richard Kuhn, et al. *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

[7] *SchemDraw*. URL: `https://schemdraw.readthedocs.io/en/latest/index.html`.

[8] *Acess Control Policy Tool*. URL: `https://csrc.nist.gov/Projects/Access-Control-Policy-Tool/ACPT`.

[9] *ACRLCS*. URL: `https://csrc.nist.gov/projects/access-control-policy-tool/access-control-rule-logic-circuit-simulation`.

[10] Anna Lisa Ferrara, P Madhusudan, and Gennaro Parlato. "Policy analysis for self-administrated role-based access control". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2013, pp. 432–447.

[11] Anna Lisa Ferrara, P Madhusudan, and Gennaro Parlato. "Security analysis of role-based access control through program verification". In: *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE. 2012, pp. 113–125.

[12] Michele Bugliesi et al. "Gran: Model checking grsecurity RBAC policies". In: *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE. 2012, pp. 126–138.

[13] Karthick Jayaraman et al. "Automatic error finding in access-control policies". In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 163–174.

[14] Evan Martin et al. "Assessing quality of policy properties in verification of access control policies". In: *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE. 2008, pp. 163–172.

[15] Mikhail I Gofman et al. "Rbac-pat: A policy analysis tool for role based access control". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2009, pp. 46–49.

[16] Kathi Fisler et al. "Verification and change-impact analysis of access-control policies". In: *Proceedings of the 27th international conference on Software engineering*. 2005, pp. 196–205.

[17] Nan Zhang, Mark Ryan, and Dimitar P Guelev. "Evaluating access control policies through model checking". In: *International Conference on Information Security*. Springer. 2005, pp. 446–460.

[18] Alessandro Cimatti et al. "Nusmv 2: An opensource tool for symbolic model checking". In: *International conference on computer aided verification*. Springer. 2002, pp. 359–364.

[19] *IBM Tivoli Security Policy Manager*. URL: `https://www.ibm.com/docs/en/datapower-gateway/10.0.1?topic=integration-tivoli-security-policy-manager`.

[20] Graham Hughes and Tevfik Bultan. "Automated verification of access control policies using a SAT solver". In: *International journal on software tools for technology transfer* 10.6 (2008), pp. 503–520.

[21] Vladimir Kolovski, James Hendler, and Bijan Parsia. "Analyzing web access control policies". In: *Proceedings of the 16th international conference on World Wide Web*. 2007, pp. 677–686.

[22] Scott D Stoller et al. "Symbolic reachability analysis for parameterized administrative role based access control". In: *Proceedings of the 14th ACM symposium on Access control models and technologies*. 2009, pp. 165–174.

[23] Somesh Jha et al. "Towards formal verification of role-based access control policies". In: *IEEE Transactions on Dependable and Secure Computing* 5.4 (2008), pp. 242–255.

[24] Frode Hansen and Vladimir Oleshchuk. "Conformance checking of RBAC policy and its implementation". In: *International Conference on Information Security Practice and Experience*. Springer. 2005, pp. 144–155.

[25] Jayalakshmi Balasubramaniam and Philip WL Fong. "A white-box policy analysis and its efficient implementation". In: *Proceedings of the 18th ACM symposium on Access control models and technologies*. 2013, pp. 149–160.

[26] Peter Amthor, Winfried E Kühnhauser, and Anja Pölck. "Heuristic safety analysis of access control models". In: *Proceedings of the 18th ACM symposium on Access control models and technologies*. 2013, pp. 137–148.

[27] Ping Yang et al. "Policy analysis for administrative role based access control without separate administration". In: *Journal of Computer Security* 23.1 (2015), pp. 1–29.

[28] Dianxiang Xu et al. "A model-based approach to automated testing of access control policies". In: *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. 2012, pp. 209–218.

[29] Alexander Pretschner, Tejeddine Mouelhi, and Yves Le Traon. "Model-based tests for access control policies". In: *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE. 2008, pp. 338–347.

[30] Hongxin Hu and GailJoon Ahn. "Enabling verification and conformance testing for access control model". In: *Proceedings of the 13th ACM symposium on Access control models and technologies*. 2008, pp. 195–204.

[31] Shinji Kikuchi et al. "Policy verification and validation framework based on model checking approach". In: *Fourth International Conference on Autonomic Computing (ICAC'07)*. IEEE. 2007, pp. 1–1.

[32] Muhammad Aqib and Riaz Ahmed Shaikh. "A tool for access control policy validation". In: *Journal of Internet Technology* 19.1 (2018), pp. 157–166.

[33] Vincent C Hu et al. "Model checking for verification of mandatory access control models and properties". In: *International Journal of Software Engineering and Knowledge Engineering* 21.01 (2011), pp. 103–127.

[34] Nan Zhang, Mark Ryan, and Dimitar P Guelev. "Synthesising verified access control systems through model checking". In: *Journal of Computer Security* 16.1 (2008), pp. 1–61.

[35] Andreas Schaad, Volkmar Lotz, and Karsten Sohr. "A model-checking approach to analysing organisational controls in a loan origination process". In: *Proceedings of the eleventh ACM symposium on Access control models and technologies*. 2006, pp. 139–149.

[36] Manuel Koch, Luigi V Mancini, and Francesco Parisi-Presicce. "Conflict detection and resolution in access control policy specifications". In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2002, pp. 223–238.

[37] Jonathon E Tidswell and Trent Jaeger. "Integrated constraints and inheritance in DTAC". In: *Proceedings of the fifth ACM workshop on Role-based access control*. 2000, pp. 93–102.

[38] Antonios Gouglidis, Ioannis Mavridis, and Vincent C Hu. "Verification of secure inter-operation properties in multi-domain RBAC systems". In: *2013 ieee seventh international conference on software security and reliability companion*. IEEE. 2013, pp. 35–44.

[39] *Multi-input XOR Interpretations*. URL: `https://mindhunter74.wordpress.com/2011/04/25/xor-the-interesting-gate/`.

A Designed Rulesets

CI.txt

```
(should fail due to cyclic inheritance)
-----------------------------------------
--------------- Attributes ---------------
-----------------------------------------
Reviewer
Editor
Designer
Admin


-----------------------------------------
------ Attribute Links / Hierarchy -------
High level attribute | Low level attribute
---------------- Optional -----------------
-----------------------------------------
Reviewer | Editor
Editor   | Designer


-----------------------------------------
--- Subjects | Inheritors | Attributes ---
-----------------------------------------
Alice   | Charlie, Derek     | Reviewer
Bob     | Charlie, Derek     | Reviewer
Charlie | Erin, Frank        | Editor
Derek   | Alice, Erin, Frank | Editor     --CYCLIC INHERITANCE
Erin    | Gary               | Designer
Frank   | Gary               | Designer
Gary    | none               | Admin


-----------------------------------------
----- Objects | Conflicting objects ------
-----------------------------------------
All pages   | OS pages
All folders | OS folders
Widgets     | None


-----------------------------------------
---------------- Actions ----------------
-----------------------------------------
View
Create
Upload
Delete
Publish
Modify

-----------------------------------------
 Attribute | Object | Action | Attributes
-----------------------------------------
```

```
Gary           | All pages, All folders, Widgets | Modify, Publish, Upload,
↪  Create, View, Delete | Admin
Designer       | OS pages, OS folders            | Publish, Create, Delete
↪                    |
Designer       | Widgets                         | Modify, Publish, Upload,
↪  Create, View         |
Editor         | OS pages, OS folders            | Modify, Upload
↪                    |
Reviewer       | OS pages, OS folders            | View
↪                    |
```

No_Fault.txt

```
(No Faults)
-----------------------------------------
--------------- Attributes ---------------
-----------------------------------------
Reviewer
Editor
Designer
Admin


-----------------------------------------
------ Attribute Links / Hierarchy -------
High level attribute | Low level attribute
--------------- Optional ----------------
-----------------------------------------
Reviewer | Editor
Editor   | Designer


-----------------------------------------
--- Subjects | Inheritors | Attributes ---
-----------------------------------------
Alice   | Charlie, Derek     | Reviewer
Bob     | Charlie, Derek     | Reviewer
Charlie | Erin, Frank        | Editor
Derek   | Erin, Frank        | Editor
Erin    | Gary               | Designer
Frank   | Gary               | Designer
Gary    | none               | Admin


-----------------------------------------
----- Objects | Conflicting objects ------
-----------------------------------------
All pages   | OS pages
All folders | OS folders
Widgets     | None


-----------------------------------------
---------------- Actions ----------------
-----------------------------------------
```

```
View
Create
Upload
Delete
Publish
Modify

-----------------------------------------
 Attribute | Object | Action | Attributes
-----------------------------------------
Gary           | All pages, All folders, Widgets | Modify, Publish, Upload,
↪  Create, View, Delete | Admin
Designer       | OS pages, OS folders            | Publish, Create, Delete
↪                    |
Designer       | Widgets                         | Modify, Publish, Upload,
↪  Create, View         |
Editor         | OS pages, OS folders            | Modify, Upload
↪                    |
Reviewer       | OS pages, OS folders            | View
↪                    |

-----------------------------------------
--------------- WORKFLOW -----------------
-----------------------------------------
(Designer | Create, Upload) -> (Reviewer | View) -> (Editor | Modify) -> (Admin
↪  | Publish)

-----------------------------------------
------- GENERAL N-PERSON CONTROL ---------
-----------------------------------------
Designer, Reviewer | Upload
```

PE.txt

```
(should fail due to privilege escalation fault)
-----------------------------------------
--------------- Attributes ---------------
-----------------------------------------
Reviewer
Editor
Designer
Admin


-----------------------------------------
------ Attribute Links / Hierarchy -------
High level attribute | Low level attribute
---------------- Optional -----------------
-----------------------------------------
Reviewer | Editor
Editor   | Designer


-----------------------------------------
--- Subjects | Inheritors | Attributes ---
```

```
-----------------------------------------
Alice   | Charlie, Derek     | Reviewer
Bob     | none               | Reviewer
Charlie | Erin, Frank        | Editor
Derek   | Erin, Frank        | Editor
Erin    | Gary               | Designer
Frank   | Gary               | Designer
Gary    | Bob                | Admin      --PRIVILEGE ESCALATION


-----------------------------------------
----- Objects | Conflicting objects ------
-----------------------------------------
All pages   | OS pages
All folders | OS folders
Widgets     | None


-----------------------------------------
---------------- Actions ----------------
-----------------------------------------
View
Create
Upload
Delete
Publish
Modify

-----------------------------------------
 Attribute | Object | Action | Attributes
-----------------------------------------
Gary           | All pages, All folders, Widgets | Modify, Publish, Upload,
↪  Create, View, Delete | Admin
Designer       | OS pages, OS folders            | Publish, Create, Delete
↪                  |
Designer       | Widgets                         | Modify, Publish, Upload,
↪  Create, View         |
Editor         | OS pages, OS folders            | Modify, Upload
↪                  |
Reviewer       | OS pages, OS folders            | View
↪                  |
```

COI.txt

```
(Fails due to Conflict of Interest)
-----------------------------------------
--------------- Attributes ---------------
-----------------------------------------
Developer
Transaction Specialist
Asset Specialist
Project Manager

-----------------------------------------
--- Subjects | Inheriting | Attributes ---
```

```
-----------------------------------------
Brett        | none                    | Developer
Caleb, Chris | none                    | Transaction Specialist
Dan          | none                    | Asset Specialist
Eric         | Alex, Brett, Caleb, Dan | Project Manager


-----------------------------------------
----- Objects | Conflicting objects ------
-----------------------------------------
Source code
Employee database
Transaction database | Asset database
Asset database | Transaction database


-----------------------------------------
---------------- Actions ----------------
-----------------------------------------
Read
Write
Add
Delete

-----------------------------------------
- Subject | Object | Action | Attributes -
-----------------------------------------
Brett | Source code                                              | Read, Write,
↪  Add, Delete | Developer
Caleb | Transaction database                                     | Read, Write,
↪  Add, Delete | Transaction Specialist
Dan   | Asset database                                           | Read, Write,
↪  Add, Delete | Asset Specialist
Eric  | Transaction Database, Asset database, Employee database | Read, Write,
↪  Add, Delete | Project Manager  (fault here)
Eric  | Source code                                              | Read
↪             | Project Manager
```

GnPC.txt

```
(Fails due to N-Person control)
-----------------------------------------
--------------- Attributes ---------------
-----------------------------------------
Junior Developer
Senior Developer
Frontend Developer
Backend Developer
Server Admin
CTO


-----------------------------------------
--- Subjects | Inheriting | Attributes ---
-----------------------------------------
```

```
Alex, Aaron | none                              | Junior developer, Frontend
↪  developer
Brayden     | Alex, Aaron                       | Senior developer, Frontend
↪  developer
Cooper          | none                              | Junior developer, Backend
↪  developer
Dylan       | Cooper                            | Senior developer, Backend
↪  developer
Eli             | none                              | Server admin
Finn            | Eli, Dylan, Cooper, Brayden, Alex | CTO



-----------------------------------------
----- Objects | Conflicting objects ------
-----------------------------------------
Client-side code (test)
Client-side code (prod)
Server-side code (test)
Server-side code (prod)
Proprietary SDK
Product database
User database
Employee database



-----------------------------------------
---------------- Actions ----------------
-----------------------------------------
Read
Write
Add
Delete


-----------------------------------------
---------------- Grant ------------------
- Subject | Object | Action | Attributes -
-----------------------------------------
Alex    | Client-side code (test)
↪                                                                    |
↪  Read, Write, Add, Delete
Alex    | Client-side code (prod), Proprietary SDK
↪                                                                   | Read
Brayden | Client-side code (test & prod)
↪                                                                   | Read,
↪  Write, Add, Delete
Brayden | Proprietary SDK
↪                                                                    | Read
Cooper  | Server-side code (test)
↪                                                                    |
↪  Read, Write, Add, Delete
Cooper  | Product/User databases, Proprietary SDK
↪                                                                    | Read
Dylan   | Server-side code (test & prod), Product/User databases
↪                                  | Read, Write, Add, Delete
```

```
Dylan   | Proprietary SDK
↪
↪                                                    | Read
Eli     | Employee database
↪    | Read, Write, Add, Delete
Finn    | Client/Server-side code (test & prod), Product/User/Employee
↪  databases  | Read
Finn    | Proprietary SDK
↪    | Read, Write, Add, Delete


-----------------------------------------
----------------- Deny ------------------
- Subject | Object | Action | Attributes -
-----------------------------------------
Junior developer | Client-side code (prod), Server-side code (prod),
↪  Proprietary SDK  | Write, Add, Delete


-------------------------------------------------
------------------ WORKFLOW --------------------
 ... -> (Attributes | Object | Actions) -> ...
-------------------------------------------------
(Junior developer, Frontent developer | Client-side code (test) | Write)
        -> (Senior developer, Frontend developer | Client-side code (test &
        ↪  prod) | Read, Write, Add, Delete)
(Junior developer, Backend developer | Server-side code (test) | Write)
        -> (Senior developer, Backend developer | Server-side code (test &
        ↪  prod) | Read, Write, Add, Delete)
(Senior developer | Proprietary SDK | Read)
        -> (CTO | Proprietary SDK | Write, Add, Delete)
(CTO -> | Employee database | Read)
        -> (Server admin | Employee database | Write, Add, Delete)


-------------------------------------------------------------------------
------------------------ General Mutual Exclusion ----------------------
---- (Attribute | Object | Action) <> (Attribute | Object | Action) -----
-------------------------------------------------------------------------
(CTO | Employee database | Read) <> (Server admin | Employee database | Write)
(Junior developer, Backend developer | Product/User databases | Read) <>
↪  (Senior developer, Backend developer | Product/User databases | Write)


(these N-person control rules should (hopefully) enable a fault condition,
since Junior devs are denied access to prod branch, but this rule would allow
↪  them access together)
-------------------------------------------------------------------------
--------------------- General N-Person Control --------------------------
----------------- Attributes | Object | Actions | N ---------------------
-------------------------------------------------------------------------
Junior developer, Frontend developer | Client-side code (prod) | Write | 2
Junior developer, Backend developer  | Server-side code (prod) | Write | 2
```

big_policy.txt

```
-----------------------------------------
--------------- Attributes --------------
-----------------------------------------
Tester
UX/UI designer
Data scientist
Junior frontend dev
Junior backend dev
Senior frontend dev
Senior backend dev
Server admin
Lead engineer
Project manager
CTO


-----------------------------------------
--- Subjects | Inheriting | Attributes ---
-----------------------------------------
Abe, Anthony, Alicia    | None                                          |
↪  Tester
Barbara, Ben, Brody     | None                                          |
↪  UX/UI designer
Carmen, Chris           | None                                          |
↪  Junior frontend dev
Denice, Daniel          | None                                          |
↪  Junior backend dev
Edison, Eric, Erica     | Carmen, Chris                                 |
↪  Senior frontend dev
Frank, Freeman, Forrest | Denice, Daniel                                |
↪  Senior backend dev
Gwen                    | None                                          |
↪  Data scientist
Hannah                  | Gwen, Frank, Freeman, Forrest, Edison, Eric, Erica |
↪  Project manager
Isaac                   | Hannah                                        |
↪  Director
Jill                    | Isaac                                         |
↪  CTO


-----------------------------------------
----- Objects | Conflicting objects ------
-----------------------------------------
Alpha version
Client-side code (test)
Client-side code (prod)
Server-side code (test)
Server-side code (prod)
Proprietary SDK
Product database
User database
Interaction database
Transaction database
```

```
Ticket database
Employee database
Master database
ML models
Backup server
API Keys
Cloud environment



-----------------------------------------
---------------- Actions ----------------
-----------------------------------------
Read
Write
Add
Delete


-----------------------------------------
- Attribute | Object | Action | Subjects -
-----------------------------------------
Tester              | Alpha version
↪
↪                 | Read
Tester              | Ticket database
↪
↪                 | Write
UI/UX Designer      | Alpha version
↪
↪                 | Read
UI/UX Designer      | Ticket database
↪
↪                 | Write
Data scientist      | Interaction database
↪
↪                 | Read
Data scientist      | ML models
↪
↪                 | Read, Write, Add, Delete
Junior frontend dev | Client-side code (test)
↪
↪                 | Read, Write, Add, Delete
Junior frontend dev | Client-side code (prod), Proprietary SDK, Alpha version
↪
↪                 | Read
Junior backend dev  | Server-side code (test)
↪
↪                 | Read, Write, Add, Delete
Junior backend dev  | Product/User/Interaction/Transaction/Ticket/Employee
↪  databases, Proprietary SDK, Alpha version, Cloud environment
↪                     | Read
Senior frontend dev | Client-side code (test & prod)
↪
↪                 | Read, Write, Add, Delete
Senior frontend dev | Alpha version, Proprietary SDK
↪
↪                 | Read
```

```
Senior backend dev  | Server-side code (test & prod),
↪  Product/User/Interaction/Ticket/Master databases, Cloud environment
↪                                          | Read, Write, Add, Delete
Senior backend dev  | Alpha version, Employee database, Proprietary SDK
↪
↪                | Read
Server admin        | Employee/Ticket databases, Backup server
↪
↪                | Read, Write, Add, Delete
Lead engineer       | Client-side code (test & prod), Server-side code (test &
↪  prod), ALL databases, Backup Server, Cloud environment
↪                 | Read, Write, Add, Delete
Lead engineer       | Proprietary SDK, API keys
↪
↪                | Read
Project manager     | Backup server, Cloud environment, Proprietary SDK, API
↪  keys
↪                  | Read, Write, Add, Delete
Project manager     | Alpha version, Client-side code (test & prod),
↪  Server-side code (test & prod), ALL databases, ML models, Backup server,
↪  API keys, Cloud environment | Read
CTO                 | EVERYTHING
↪
↪                | Read, Write, Add, Delete



-----------------------------------------------
------------------ WORKFLOW --------------------
 ... -> (Attributes | Object | Actions) -> ...
-----------------------------------------------
(UI/UX designer | Alpha version | Read)
    -> (Server admin | Ticket database | Add, Delete)
    -> (Junior frontend dev | Client-side code (test) | Read, Write, Add, Delete)
(Tester | Alpha version | Read)
    -> (Server admin | Ticket database | Add, Delete)
    -> (Junior frontend/backend dev | Client-side code (test) | Read, Write,
    ↪  Add, Delete)
(Junior frontend dev | Client-side code (test) | Write)
        -> (Senior frontend dev | Client-side code (test & prod) | Read, Write,
        ↪  Add, Delete)
(Junior backend dev | Server-side code (test) | Write)
        -> (Senior backend dev | Server-side code (test & prod) | Read, Write,
        ↪  Add, Delete)
(Senior developer | Proprietary SDK | Read)
        -> (CTO | Proprietary SDK | Write, Add, Delete)
(Project manager | Interaction database | Read)
    -> (Data scientist | ML Models | Read, Write, Add, Delete)
(Lead engineer | Backup server | Read)
    -> (Server admin | Backup server | Read, Write, Add, Delete)
(Project manager | Employee database | Read)
    -> (CTO | Employee database | Read)
        -> (Server admin | Employee database | Write, Add, Delete)


-------------------------------------------------------------------------
```

```
------------------------ General Mutual Exclusion ----------------------
---- (Attribute | Object | Action) <> (Attribute | Object | Action) -----
------------------------------------------------------------------------
(CTO | Employee database | Read) <> (Server admin | Employee database | Write)
(Junior backend dev | Product/User databases | Read) <> (Senior backend dev |
↪  Product/User databases | Write)


------------------------------------------------------------------------
--------------------- General N-Person Control --------------------------
----------------- Attributes | Object | Actions | N ---------------------
------------------------------------------------------------------------
Senior backend dev | API Keys | Write | 3
```