CISC 835 An Analysis of Alloy in Cybersecurity Through Modeling of The Android Permissions System

Nicholas Dionne (24kg1@queensu.ca)

1 INTRODUCTION

This paper dives into Formal Methods in Cybersecurity; after a brief amount of background information on Formal Methods and the utilization of Formal Methods in Cybersecurity, this paper then introduces Alloy as a formal specification language and covers issues found in the Android permissions system as grounds for the application of Alloy, and it's further analysis as a Formal Methods tool utilized for the identification of bugs and or flaws to secure and verify the security of systems. Some questions this paper will attempt to answer and consider are: Are Formal Methods a viable option for helping secure systems in Cybersecurity? Is Alloy as a Formal Methods tool a viable, scalable, accurate, and or practical means of locating flaws and securing systems in the case of the Android permissions system. The paper will consider these questions and more by analyzing the corresponding referenced papers and the author's first-hand experience in the replication and implementation of Alloy, a Formal Methods tool, to identify and secure flaws in the Android permissions system.

2 BACKGROUND

The subsections on Formal Methods and Formal Methods in Cybersecurity are background information presented to help provide a more comprehensive understanding of the following sections on the use of Alloy in Cybersecurity analysis, specifically Android security. These papers are published by NIST (National Institute of Standards and Technology) and IEEE (The Institute of Electrical and Electronics Engineers): *What Happened to Formal Methods for Security?* and *Insights on Formal Methods in Cybersecurity*, along with the *Formal Methods 835 course notes*, will be utilized to form this understanding [3, 4, 6].

2.1 Formal Methods

What are Formal Methods? Formal Methods are "techniques and tools that use formal specification lan-guages" [3]. These tools can be utilized to improve the software development process and the subsequent resulting software [3]. Formal Methods, in all actuality, are a standardized methodology for thinking on and modeling high-level ideas in software; this method can then be used due to its well-defined and uniquely unambiguous properties to model and test the specifications of a system [3]. Formal Methods often do this through tools and methods such as Alloy, the focus of this paper [3]. Alloy is a formal specification language for modeling object-oriented systems, the relations between said objects, and the operations that can be performed on those objects [3].

2.2 Cybersecurity

2.2.1 Formal Methods in Cybersecurity. Formal Methods, as previously outlined, use mathematical and logical techniques to design, specify, and verify software and hardware systems [3]. Formal Methods offer a rigorous assurance that systems will function as intended and securely, especially in safety-critical domains [3]. Ensuring the quality of safety-critical systems, this area is where Formal Methods seem to have made notable achievements, and this idea of a methodology for ensuring the quality of a system is seemingly perfect for use in Cybersecurity [4, 6]. In fact, Formal Methods are seeing an overall increase in use across Cybersecurity; some use cases include how Formal Methods are utilized in the design and analysis of security-critical software systems by formal modeling and interactive theorem proving [4, 6]. Static analysis tools that also utilize fundamental principles from Formal Methods analyze source code for vulnerabilities without any execution, and these tools will typically automatically detect bugs like buffer overflows, memory leaks, and even race conditions [4, 6].

2.2.2 Benefits of Formal Methods in Cybersecurity. Moreover, there are some outstanding benefits to the further introduction of Formal Methods in Cybersecurity; Formal Methods offer a proactive approach to security by helping to identify vulnerabilities and design flaws early in the development life-cycle by helping to shift from today's typical reactive security notion of "build it, hack it, patch it" [4] to a more advanced approach [4, 6]. Additional benefits of Formal Methods in Cybersecurity include the following: high assurance and reduced costs [4, 6]. High assurance simply means using mathematical proof to demonstrate that system security requirements are met; this can show the absence of security bugs and or flaws rather than as in testing, where it can only detect the presence of those bugs, not their absence [4, 6]. The idea of Formal Methods reducing costs is due to the fact that although there may be an upfront cost, the use of Formal Methods to catch flaws and or bugs in implementation and design early on in development will pay for itself as it naturally minimizes the need for extensive testing, bug fixes, and security patches at a later date [4, 6].

2.2.3 Challenges of Formal Methods in Cybersecurity. However, the two previously mentioned articles on Formal Methods in Cybersecurity note some challenges. The more notable of these challenges are automation and integration with existing practices, all of which we may see in further detail and with more concrete evidence when later analyzing Alloy in Android security [4, 6]. These are defined as follows: automation of Formal Methods has seen significant progress, but unfortunately, it still requires a considerable amount of manual effort, particularly during the process of creating formal specifications; integration is the idea that for broader adoption of Formal Methods as a security and design tool, tools that are user friendly and require the least amount of extensive knowledge in Formal Methods are needed [4, 6].

2.2.4 Final Thoughts on Formal Methods in Cybersecurity. Finally, Formal Methods offer a rigorous and systematic way to design, analyze, and verify the security of systems in Cybersecurity. Challenges aside, the increasing complexity of cyber attacks and the growing need for high-assurance systems seemingly make Formal

Methods a crucial tool in the ongoing battle to build more secure hardware and software.

3 ANDROID

The following papers that will be referenced to improve the understanding of permission systems in Android and Cybersecurity are *A Formal Approach for Detection of Security Flaws in the Android Permission System*, *Resolving the Predicament of Android Custom Permissions* , and *A Comprehensive Analysis of the Android Permissions System* [1, 2, 5].

3.1 Android Permissions

Android is one of the most widely used operating systems on the globe; it provides a rather robust security model that protects sensitive user data and system resources [1, 2, 5]. The security model mainly achieves these goals through the Android permissions system, and the permissions system governs how applications access resources such as the user's location, camera, or user contacts [1, 2, 5]. This permissions system essentially has the sole duty of preventing malicious applications from exploiting vulnerabilities to gain unauthorized access to a user's sensitive and private information [1, 2, 5].

3.2 Android Permissions in Detail

In further detail, it can be seen that the Android permissions system operates on a policy-driven model, these policies are as follows [1, 2, 5]. Predefined permissions are the set of standard permissions that Android defines, such as the previously mentioned location, camera, and user contacts that applications can request for use; these policy's permissions can also be further grouped into normal and dangerous permissions; internet would be categorized as normal and camera and location as dangerous [1, 2, 5]. Custom permissions are permissions that developers can define for their applications, allowing for unique enabling of specific inter-application interactions; this policy allows for a lot of flexibility in the Android permissions system while also being a leading cause of vulnerabilities when those custom permissions are poorly configured [1, 2, 5]. URI permissions are rather special; they're used to control access to content URI's, which define database entries and files; these permissions allow applications to share specific data relatively securely utilizing a content provider, URI permissions are also either temporary or persistent [1, 2, 5]. However, improperly managed or exploited URI permissions can allow other unauthorized applications to access and manipulate potentially sensitive data [1, 2, 5]. Runtime permissions, which were introduced from Android 6.0, are a permissions policy where permissions that are classified as dangerous are no longer directly granted to applications after installation as was standard before the introduction of runtime permissions; instead, users now have to approve the use of those dangerous permissions during application runtime which fortunately provides greater control to the user but adds an additional set of dynamic behaviors to model which may complicate an already complex analysis of the Android permissions system [1, 2, 5].

3.3 Android Permission System Flaws

Permissions systems in Android or just about any OS (operating system) are essential for security against applications [1, 2, 5]. In Cybersecurity, permission systems maintain security by following the principle of least privilege, meaning that applications should have access only to the resources necessary in order to have complete functionality [1, 2, 5]. This design, in concept, should completely prevent any outstanding vulnerabilities; however, there are flaws in every design in this particular instance; some applications may request more permissions than required for functionality, which subsequently creates new attack vectors [1, 2, 5]. Moreover, there is also improper delegation, where some applications with access to sensitive data may unintentionally misplace that data into the hands of other applications [1, 2, 5]. Lastly, as previously mentioned, in the case of Android, which allows custom permissions for applications, some developers may sometimes define their permissions too broadly, which would then expose their application to the exploitations of other applications or users [1, 2, 5].

3.4 Android Permission System Challenges

These flaws and the overall design of the Android permissions system present some complex and unique challenges for securing and analyzing the system [1, 2, 5]. Android applications often interact through intents or shared resources, which can create indirect access paths that are rather difficult to analyze or actively monitor [1, 2, 5]. The nature of the Android permissions system, especially with the addition of runtime permissions after Android 6.0, makes it considerably harder to predict application behaviors as certain dangerous permissions can now be revoked at runtime, depending entirely on user decisions and context [1, 2, 5]. Furthermore, there is the fact that Android is an actively evolving framework [1, 2, 5]. In other words, with each Android release, there is potential for the permissions system model to grow and change, which may introduce new mechanisms that require an updated method of analysis, such as when runtime permissions were added in Android 6.0, making the permissions system even more dynamic [1, 2, 5].

4 ALLOY

The following papers that will be referenced to further understand Formal Methods and the importance of Formal Methods tools, such as Alloy, for application in Cybersecurity are *A Formal Approach for Detection of Security Flaws in the Android Permission System*, *Resolving the Predicament of Android Custom Permissions* , *Formal Methods 835 course notes* [1, 3, 5].

4.1 What is Alloy?

Alloy, as previously detailed, is a formal specification language designed to model and analyze complex systems. Alloy applies the principles of Formal Methods to allow for the creation of models that help define the behavior, relationships, and constraints of a particular system, such as the Android permissions system [1, 3, 5]. Such models can then be analyzed by the Alloy analyzer, which checks the model for logical inconsistencies while generating counterexamples and validating assertions [1, 3, 5]. Alloy is also not imperative but declarative; in other words, the Alloy language allows the use of a compact and expressive syntax to describe system components and their relationships, allowing developers to focus on the properties of the system they wish to model rather than the implementation details [1, 3, 5]. Bounded model checking is a crucial aspect of Alloy; it allows for exhaustive checks within a predefined set of bounds, such as the number of objects or relations, and this bounded approach helps ensure that analysis of a system model remains computationally feasible while covering many potential specifications [1, 3, 5]. Lastly, to bring more detail to the specifics, Alloy applies Formal Methods to define system behaviors through signatures that represent entities or specific system components, facts that define global constraints that should always hold true, and predicates as well as assertions that specify a desired property and or conditions necessary to verify system correctness in the model [1, 3, 5]. These particular methods utilized by Alloy are the reason Alloy is a favored method in certain security designs, as Alloy can be used to rigorously define system behaviors, therefore detecting vulnerabilities and validating security policies such as permissions policies [1, 3, 5].

4.2 Alloy Strengths

Some of the identifiable strengths of Alloy that may garner its use no matter the setting include the following. Alloy excels in precision and early detection by locating logical inconsistencies and design flaws before implementation [1, 3, 5]. This can help give a modeled system a high level of assurance and reduce the overall cost by minimizing potential issues with design in the early stages, as previously mentioned [1, 3, 5]. Counterexamples generated automatically through violated assertions that are detected by the Alloy analyzer provide developers with the clear guidance needed on how to address certain vulnerabilities in the design [1, 3, 5]. One of the critical issues with Formal Methods adoption in Cybersecurity is the accessibility and usability of Formal Methods tools, Alloy unlike other Formal Methods tools has a relatively easy-to-learn syntax, which makes it more accessible to developers with security backgrounds that lack extensive Formal Methods expertise; this strength due to Alloy's design allows for further adoption and application of Formal Methods in securing systems [1, 3, 5]. The declarative nature of Alloy also allows for the modeling of a broader range of systems, from access control policies such as the Android permissions system to more complex software architectures [1, 3, 5].

4.3 Alloy Weaknesses

Alloy has as many weaknesses as it does strengths; of these weaknesses, scalability is the most troublesome for wider adoption of Formal Methods in security. The very nature of Alloy, which includes bounded verification, limits Alloy's ability to scale to large or highly dynamic systems as analysis becomes increasingly computationally expensive as the number of elements or relationships in a system increases [1, 3, 5]. This inevitably impairs Alloy's practical application for modern versions of the Android OS with its never-ending growth and evolving features [1, 3, 5]. Furthermore, Alloy struggles with dynamic behaviors, such as with Android 6.0's runtime permissions policy, which adds a significant dynamic element to the Android permissions system [1, 3, 5]. This new level of unpredictability is something that Alloy's more static models can not yet entirely address [1, 3, 5]. A rather problematic area for Alloy is that the effectiveness of Alloy is directly tied to the model quality; in other words, if the developer lacks in any way an adequate amount of understanding of the system their modeling, they may induce system behaviors and constraints that are not appropriately defined which will cause the subsequent analysis to potentially overlook critical vulnerabilities [1, 3, 5]. In fact, it could be said that the potential for human error directly counters the advantages of not needing to provide a developer with extensive Formal Methods expertise due to Alloy's simple syntax. Finally, Alloy provides limited automation in the Alloy analyzer and Alloy's simplification of the modeling process through its forgiving syntax; however, Alloy still requires a manual effort to define those ever-important system properties and constraints that are susceptible to error and, therefore, as of now, Alloy currently falls short in automation for creating specifications [1, 3, 5]. This lack of automation for specifications presents an issue for the Cybersecurity adoption of Alloy as OS's such as Android and even applications are consistently receiving updates and patches for new attacks and features [1, 3, 5]. Although methods such as Alloy may considerably reduce the initial issues when releasing software due to the high assurance provided by testing with Formal Methods, it's unlikely that tools such as Alloy in their current state could keep up with this constant demand [1, 3, 5].

5 IMPLEMENTATION

Alloy is a rather robust tool that can be used for enhancing Cybersecurity practices in software through modeling and analysis. Utilizing Formal Methods tools such as Alloy, security specialists will be able to model the architecture and interactions in a system to detect any vulnerabilities at the early design stage and validate the absence of any security bugs and that any desired security policies meet certain criteria [1, 5]. Some key points for the implementation of Alloy in Cybersecurity may be in modeling and analyzing protocols, general system design, or access control policies to ensure the enforcement of security principles like least privilege and separation of duties as in the Android permissions system, which I will lightly replicate and implement as described in *A Formal Approach for Detection of Security Flaws in the Android Permission System*, while further utilizing *Resolving the Predicament of Android Custom Permissions* for insights into methodology and implementation of Alloy for modeling and analyzing the Android permissions system [1, 5].

5.1 First-Hand Experience With Alloy for Android Permissions

The Android permissions system makes for a challenging study of Alloy's capability as a tool for securing systems due to the evolving complexity and dynamic behaviors that are introduced. Utilizing the papers, I replicated the corresponding Alloy code and results as part of my exploration of Alloy; this model formalizes the Android permissions system by defining key components, permissions, and interactions between applications and system elements. The Alloy model consists of several key elements, such as signatures, which represent the core entities in the system like Device, Application, Permission, and Component [1]. As an example, the signature Application includes some fields for the declaredPerms which are permissions declared by the application (custom permissions), usesPerms which are permissions requested by the application, and grantedPerms which are permissions granted to the application at runtime [1]. The facts and predicates include system-wide constraints and operations such as install which indicates application installation, grantPermissions which grants permissions to an installed application and invoke which is the invocation of components; these predicates combined model the cycle of permissions and general interactions in the Android permissions system [1].

Some of the model assertions involved are UniquePermissionName, NoUnauthorizedAccess, NoPrivateContentAccessedByExternalApp, and NoImproperDelegation [1]. UniquePermissionName ensures that no two permissions share the same name; however, a counterexample introduced by the Alloy Analyzer indicates otherwise; this points to a potential flaw in the model constraints or in how permissions can sometimes be defined by the Android permissions system [1]. NoUnauthorizedAccess verifies that applications will not be able to access resources without authorization. This was, in fact, also violated as the Alloy analyzer was able to find a few specific counterexamples where unauthorized access to resources was possible due to a lack of constraints in the delegation of permissions or the invocation of rules [1]. Following this, we have NoPrivateContentAccessedByExternalApp, which was also violated in some cases where the Alloy analyzer found that supposedly private content paths could be accessed by unauthorized applications under certain conditions, which indicates that URI permissions are not being effectively enforced or are being circumvented [1]. Lastly, NoImproperDelegation checks that only authorized delegation of permissions takes place, but this was also found to be violated, therefore allowing a component to indirectly authorize the access of another component without the prerequisite authorization [1].

During my re-implementation of the model, I wanted to effectively identify as well as assess the researcher's approach to modeling runtime permissions, which seem inherently problematic for Alloy due to its dynamic behavior. I thought this would be an overall weakness for the application of Alloy as a security tool, as many vulnerabilities are taken advantage of by more dynamic influences where the attackers make varying inputs or approaches to attack a vulnerability and exploit it. The result is that predefined and custom permissions were effectively modeled, but runtime permissions had to be handled through some approximations due to their dynamic behavior. To be more specific, an extension of the model was introduced to handle this dynamic behavior by allowing for the granting or revocation of permissions at each step in the Alloy analyzer to simulate the ability of a user to grant or revoke dangerous predefined permissions during runtime [1]. Although the researchers did ultimately handle the runtime permissions within Alloy, this addition added noticeable issues to the question of scalability, which was already quite strained as the model had to be purposefully simplified in certain aspects, such as the more intricate parts of the permissions themselves, as the increasing number of components, permissions, and applications significantly grew the analysis time [1]. This simplification could have ultimately compromised or limited the capability of the model to capture and simulate real-world scenarios in which the actual vulnerabilities in the design matter.

Lastly, I want to touch upon custom permissions as it's rather interesting how this paper [1] simplifies the Alloy model with regards to each permission when compared to the extensive Android custom permissions Alloy model found in this [5] paper [1, 5]. The general Android permissions model did cover custom permissions, and in fact, they found some problematic instances, which I had a look at in my re-implementation, such as custom permissions allowing for two permissions to have the same name and a possibility of dangling permissions remaining after the de-installation of an application [1]. The detailed Alloy model of the custom permissions found the exact same issues and nothing more that was particularly outstanding, which shows that even a less detailed and general model in Alloy can be quite useful, the issue being that if experts want to fix the issues they find in there design counterexamples provided by a general model may not give sufficient detail to permanently resolve the issues [5]. However, finely detailing every aspect of a system as extensive as the Android permissions system can quickly become computationally expensive, time-consuming, and affect the overall scalability of Alloy. Alloy is, however, even when used generally to model a simplified system, quite useful from my experience as it can provide that high assurance of a lack of flaws in the foundation of your design and logic.

5.2 Alloy's Role in Strengthening Android Security

By re-implementing the model and design found in the paper [1] I was able to confirm by utilizing the Alloy analyzer some of the key vulnerabilities identified in the paper. Such as URI permission flaws through the NoPrivateContentAccessedByExternalApp assertion and improper delegation, which was proven through the assertion NoImproperDelegation [1]. These two flaws present rather serious issues for Android security and validate the utility of Alloy as a security tool for checking a design for any underlying flaws, as improper delegation can unintentionally allow applications to delegate permissions to other applications, and improperly granted URI permissions can expose rather sensitive data to unauthorized applications [1]. These counterexamples, generated by the Alloy analyzer as I see it, provide actionable insight into the Android permissions systems vulnerabilities, which can allow for clear fixes or adjustments to the design if need be, and if not a clear fix then due to a lack of detailed information Alloy can at least provide a fundamental understanding that there is an issue present in a certain portion of the design. The detection of these design-level flaws before implementation can effectively reduce reactive patching in Cybersecurity by instead introducing a more proactive approach that more thoroughly verifies a system before its deployment.

6 EFFICACY

This section will utilize these titles as well as previously outlined details in this paper: *A Formal Approach for Detection of Security Flaws in the Android Permission System*, and *Resolving the Predicament of Android Custom Permissions* to take a further and more comprehensively detailed look at the efficacy of Alloy as a tool for Cybersecurity utilizing the Android permissions system models as a basis for this evaluation and insight and some of my first-hand experience with these models [1, 5].

6.1 Precision

Precision references Alloy's ability to detect and correctly define system vulnerabilities. As previously detailed, Alloy was able to effectively locate, even utilizing a simplified model, vulnerabilities in the Android permissions system [1]. Alloy provided counterexamples where applications accessed unauthorized content URIs, improper delegation where the Alloy analyzer found vulnerabilities in how permissions were delegated this shows cases where security can be breached due to inadequate constraints, or custom permissions which Alloy identified the potential for naming conflicts and overly broad access being granted to custom permissions both of which could be detrimental to security if exploited [1]. The precision offered by assertions in Alloy allows for these design-level vulnerability discoveries, such as the case of NoUnauthorizedAccess, which, when violated during Alloy's analysis, pointed to the gaps in permission delegation [1]. Moreover, to be more specific, this precision is due to Alloy's systematic verification of logical properties, which allows for the detection of specific cases where constraints fail, which makes Alloy particularly well suited for design-level vulnerability checks and, therefore, quite useful in early implementation of Cybersecurity to software [1].

6.2 Scalability

Scalability refers to how well Alloy is able to perform as the scope and complexity of the model grow. Both papers involving Android permission systems modeling utilizing Alloy made note of the fact that there was a considerable decrease in Alloy's performance as the number of components, permissions, and applications involved increased[1, 5]. This means that larger models lead to significantly longer analysis times, and as in this [1] paper, which attempted to capture the whole of the Alloy permissions system, simplifications to the model were required to maintain feasibility [1, 5]. In my replication of the model, this remained evident as the introduction of the runtime permission handling in their full model seemed to add considerable complexity, which could be said to have further strained Alloy's Scalability. This is due to the method utilized to handle runtime permissions, which is through an iterative granting and revocation of the permissions required by each application during each of Alloy's time steps, this requires considerably more computational resources and likely required the further simplification of the original model to keep analysis time within a feasible range [1, 5].

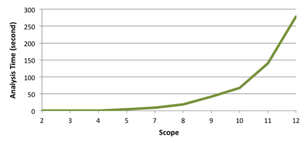

Fig. 7: Average analysis times over an increasing bound (scope) for top-level signatures

Figure 1: Fig. 7 From [1]

Figure 7 above is from *A Formal Approach for Detection of Security Flaws in the Android Permission System* which utilizes this graph to showcase the exponential increase in complexity of their model through the total Alloy analyzer analysis time as they increase the scope of there model to find further vulnerabilities [1]. Alloy, as seen in figure 7, utilizes bounded verification, which can inherently limit its scalability, making Alloy more suitable for small to medium-sized models rather than larger models with more dynamic aspects like in Android [1]. This, however, does not entirely indicate that Alloy can not be effectively utilized as the researchers, through simplification and testing of their model, found that testing past a scope of twelve in the case of their model was unlikely to present new serious vulnerabilities and was satisfactory as proof of the feasibility of using an automated verification tool on a widely used OS [1]. A case to be made, however, is the inevitable trade-off as by omitting or simplifying certain intricacies in systems such as the Android permissions system, it's likely that the subsequent analysis will lose for lack of a better words richness or detail [1]. An an example would be how interactions between multiple applications and indirect delegation paths had to be simplified to keep the analysis feasible which may allow for some inaccuracies or substance to be lost in counterexamples that would typically provide actionable results that could help patch or rethink design flaws [1].

6.3 Usability

Usability refers to how accessible Alloy is to practitioners, in this case, Cybersecurity personnel, particularly the ones without any prior ability in Formal Methods. Alloy presents a rather usable approach to Formal Methods as one of its core strengths is its declarative syntax, which allows for concise modeling of complex relationships. This enables non-experts with computer science and Cybersecurity knowledge to pick up Alloy as another tool in their security arsenal [1]. However, in my experience although, defining relationships between permissions, components, and applications seems intuitive and manageable as long as the individual responsible truly understands fully the system their modeling and its intricacies. Constructing assertions that are useful and provide proper feedback is another thing entirely, not to mention the difficulties involved with modeling dynamic behavior in Alloy while properly balancing the corresponding simplification of the model to maintain feasibility [1]. It could be said that Alloy is certainly more accessible than alot of other Formal Methods tools and is therefore a better option for Cybersecurity in some cases but its overall usability can tank when facing dynamic behavior and I believe that when rather extensive domain-specific knowledge is required of the modeler this can happen as well.

6.4 Practicality

Practicality refers to if Alloys findings are something that's actionable and can be applied to real world security scenarios. The Alloy analyzer does, in fact, generate rather clear and actionable insights as with URI permissions flaw, which directly points to specific instances of misconfigurations that a developer can then address in the Android permissions system design [1]. I found the counterexamples during my re-implementation of the model to be straightforward and relatively easy to interpret I'd imagine that this would even more so be the case if the modeler themselves, who thoroughly understands the system, were to distinguish the results. A good showcase of the practical use of Alloy is how counterexamples are, in fact, tied to real-world vulnerabilities, as the URI permissions were an actual vulnerability in the Android permissions system until a patch in Android 2.3.7 [1]. However, the only reason this vulnerability shows up in this model presented by the paper is due to trade-offs from simplifications made by the researchers during the modeling of the Android permissions system to keep the model feasible in the first place [1]. This isn't to say that nothing came of modeling the Android permissions system as it located and confirmed previously found and yet patched vulnerabilities and new vulnerabilities at the time, which are the improper delegation and custom permission flaws, respectively [1]. That aside, Alloy's focus on design-level flaws certainly limits its scope of operation in a Cybersecurity environment, as it seems to be a generally less effective method for identifying vulnerabilities that show during runtime; this is the reason, from my understanding as to why tools such as Alloy are typically paired with other tools such as static or dynamic analysis tools which are particularly well suited to the task of analyzing runtime behaviors and therefore complementing Alloy rather well [1]. To conclude, Alloy's practicality is in its ability to identify vulnerabilities in early design phases to reduce further future costs through an extensive need for reactive patching that may lead to further issues later. Still, Alloy's limitations when it comes to handling runtime and dynamic behavior can come at a cost in precision and may require the complementary use of other tools to pick up any of the slack in its stead.

7 RESULTS

This section, in all likelihood, will utilize some if not all works so far presented in this paper and the author's own experience to formulate the results of this paper and look into any potential for future work related to Alloy and Formal Methods in Cybersecurity. The titles present in the remaining section below are as follows: *A Formal Approach for Detection of Security Flaws in the Android Permission System*, *Resolving the Predicament of Android Custom Permissions*, *A Comprehensive Analysis of the Android Permissions System*, *What Happened to Formal Methods for Security?* and *Insights on Formal Methods in Cybersecurity*, and the *Formal Methods 835 course notes* [1–6].

7.1 Significance of Alloy and Android Permissions System

Android's permissions system is rather instrumental to the security of sensitive user data and system resources. Due to the use of Android on a wide scale across the globe in billions of devices, a vulnerability inside of the permissions system could be more than simply detrimental to users, exposing them to significant risks such as data breaches and unauthorized access, each with the corresponding consequences. Formal Methods tools like Alloy have the vital role of identifying these design-level vulnerabilities that can be overlooked otherwise. Alloy provided, in the case of the Android permissions system, information on newly identified vulnerabilities in custom permissions but also confirmed prior findings of other researchers on improper delegation of permissions, which through counterexamples may present new and useful information on how to approach the rectification of this vulnerability. Formal Methods such as Alloy also present a more proactive approach to mitigating risks by modeling and testing software thoroughly before its release, providing a high level of assurance by providing proof of the absence of vulnerabilities in a design, unlike some traditional testing practices. On a broader Cybersecurity spectrum, Alloy enforces principles such as least privilege, which is vital to ensuring that applications and other mediums access only the resources necessary for their basic functionality.

7.2 Insights and Lessons Learned

Alloy is a precise tool that excels in identifying logical vulnerabilities in the way that permissions are delegated and how custom permissions have naming conflicts and overly broad capacity. Alloy has to handle runtime permissions through the use of approximations and simplifications in the model this presents Alloy's difficulties with modeling dynamic systems without trade-offs in precision and scalability. The more I looked into Alloy as a security tool, the more I found its use as a standalone tool to be problematic in the sense that Alloy requires other tools to make up for its limitations, and it would not be a stretch to say that Alloy should not be utilized beyond its purpose as a design-level vulnerability checker at least in security. Furthermore, although the researchers in this [1] paper found Alloy to be a feasible option when securing large OS's due to their testing in regards to Alloy's scope, it needs to be noted that their model has been simplified and approximated at an expense to be able to capture the model of an outdated and arguably considerably less complex Android permission system then what is present today although fundamentally quite similar in overall concept [1].

7.3 Future Work

This [1] paper makes clear their desire to, in future work, create "an end-to-end security analysis framework that combines a model-based detection of system-level attacks with a suite of static analysis tools that can identify particular types of vulnerabilities" [1] this is undoubtedly in my mind a step in the right direction for Formal Methods such as Alloy to see more use in Cybersecurity. That aside, breaking down larger models into smaller parts for incremental verification is certainly a method that could be used to avoid too many trade-offs, such as in this [5] paper where rather than modeling the entire Android permissions system, only the custom permissions were modeled in quite the detail [1, 5]. Still, ultimately, this means the model loses most of the unique interactions in the entire system between several applications and several permissions and their unique properties. Alloy could potentially be combined with some runtime analysis tools to help it better account for and handle dynamic behaviors at less of a cost to the overall model (It seems some already exist after a quick search; however, it's uncertain as to the potential trade-offs, and utility associated with these methods). Alloy could be further leveraged for use in other security-critical domains like IoT systems; this is likely a perfect fit for Alloy as the designs for IoT are often simple and small models with similar access control mechanisms to Android permissions, meaning that Alloy is unlikely to have to face many if any of its limitations here to provide effective feedback that reinforces IoT systems that are notoriously insecure.

7.4 Conclusion

This paper gave the corresponding background information and analyzed the efficacy of Alloy as a tool for Cybersecurity by examining its application to the Android permissions system through first-hand experience and insights gleaned from the scientific rigor among various referenced papers. From this analysis, it can be seen that Alloy is phenomenal at detecting design-level vulnerabilities, such as improper delegation, and issuing precise and actionable results through counterexamples. Alloy is also a tool with high usability, making it a suitable choice for Cybersecurity professionals without Formal Methods expertise, though modeling can still remain a challenge for rather complex systems, especially when a lot of dynamic behavior is involved. The scalability of Alloy can present some challenges that limit the application of Alloy to larger models; however, the potential for high assurance provided through early security verifications at the design level can offer significant advantages from the results. From the application of Alloy in the limited setting of the Android permissions system, the broader importance of Formal Methods in Cybersecurity can be seen, as Formal Methods help address the ever-growing complexity of our modern systems. Although Alloy on its own may be insufficient for complete security its approach to checking system design can certainly provide a significant boost to the security of a system in most cases.

7.5 Note

This [1] papers model can be found here https://groups.csail.mit.edu/sdg/projects/android/. Furthermore, this [5] papers model can be found here https://github.com/gulizseray/alloy-android-permissions on GitHub.

REFERENCES

[1] I. Aktas, D. Basin, M. D. Schreckling, and R. Sasse. 2017. A Formal Approach for Detection of Security Flaws in the Android Permission System. *Formal Aspects of Computing* 29, 3 (2017), 389–418. https://doi.org/10.1007/s00165-017-0445-6

[2] Iman M. Almomani and Aala Al Khayer. 2020. A Comprehensive Analysis of the Android Permissions System. *IEEE Access* 8 (2020), 216671–216688. https://doi.org/10.1109/ACCESS.2020.3041432

[3] Juergen Dingle. 2024. CISC 422/835: Formal Methods in Software Engineering. (2024). Course notes, Fall 2024, Queen's University.

[4] Kim Schaffer and Jeffrey Voas. 2016. What Happened to Formal Methods for Security? *Computer* 49, 8 (2016), 70–75. https://doi.org/10.1109/MC.2016.245

[5] Güliz Seray Tuncay, Soteris Demetriou, Karan Ganju, and Carl A. Gunter. 2018. Resolving the Predicament of Android Custom Permissions. In *Proceedings of the 25th Network and Distributed System Security (NDSS) Symposium.* Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2018.23210

[6] Jeffrey Voas, Kim Schaffer, et al. 2016. Insights on Formal Methods in Cybersecurity. *Computer* 49, 5 (2016), 102–104. https://doi.org/10.1109/MC.2016.133