CACLE - Automated Mitigation for Misconfiguration Vulnerabilities in Cloud Systems

Douglas J. Millward
University of Essex, UK

Martin J. Reed
University of Essex, UK

Nkaepe Olaniyi
MKC Training Services Limited, Royal School of Military Engineering, UK

ABSTRACT

Recent studies have highlighted the increasing risk to Cloud Systems of User Misconfiguration Errors. This paper provides an evolutionary approach to identifying and mitigating such errors, known as the Configuration and Checking Logic Engine. We demonstrate its effectiveness utilizing test files from publicly available GitHub repositories. The results show that not only is it as effective as other solutions that check IaC templates, but it is also an order of magnitude faster than existing approaches.

CCS CONCEPTS

• **Software and its engineering**; • **Software testing and debugging;**; • **Security and privacy**; • **Software and application security;**; • **General and reference**; • **Design;**; • **Computing methodologies**; • **Distributed computing methodologies;**;

KEYWORDS

Additional Key Words and Phrases: cloud security, infrastructure as code, automated vulnerability mitigation, user misconfiguration errors

ACM Reference Format:
Douglas J. Millward, Martin J. Reed, and Nkaepe Olaniyi. 2023. CACLE - Automated Mitigation for Misconfiguration Vulnerabilities in Cloud Systems. In *2023 7th International Conference on Cloud and Big Data Computing (ICCBDC 2023), August 17–19, 2023, Manchester, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3616131.3616142

1 INTRODUCTION

"Shift-left" security is a relatively recently-coined phrase popularized by the DevOps community recommending that security should be embedded into the earliest phase of the development process possible [1]. It signifies a commercial acceptance of security issues that have been known in academia for over a decade [2] that identifying and mitigating vulnerabilities in the earlier phases of the development cycle is both faster and more effective. There are an increasing number of causes of such vulnerabilities: poor or incomplete requirements, inadequate requirements engineering, and poor design being the most common and well-known. In the past few years, a new cause of security vulnerabilities for cloud computing systems has emerged - User Misconfiguration Errors (UMEs) [3]. This paper introduces the Configuration Analysis and Checking Logic Engine (CACLE) which provides a new mechanism for dealing with such errors. It also demonstrates the use of the engine on a publicly available data set.

The key to the flexibility of the cloud is the use of Infrastructure as Code (IaC) [4]. IaC has also led to the process management methodology known as DevOps [5] by providing a mechanism to automatically configure system depen- dencies and provision local and remote instances. This in turn has engendered a dependency on configuration files and mechanisms which has led to the rise of automated configuration tools such as Apache Brooklyn, Chef, Puppet, and similar applications. In many ways, these tools are double-edged swords in that they introduce new avenues for security vulnerabilities (through UMEs) as well as providing a potential channel for identifying and mitigating such vulnerabilities. Utilizing this latter aspect is the focus of this research.

One of the drivers for the increase in UMEs is the changing nature of the System Development Life Cycle [6]. The transition from more traditional paradigms such as waterfall and spiral, through agile models to modern DevOps approaches has heralded a move to ever greater automation in the development life cycle. However, along with this increase has been a commensurate decrease in the informal communication processes highlighted by [6], and that has meant a reduction in both the formal and informal security checks that such communication engenders. A study [7] of the practices and issues involved in configuration design and management summarises both the impact of UMEs on large, international companies such as Facebook and Google, as well as highlighting the critical improvements required to reduce UMEs at a systemic level. Although Zhang et al. [7] do not focus on security, there are three recommendations from their research that are appropriate to and align with the objectives of our paper:

- the provision of proactive parameter value checking and validation - we focus specifically on **security**
- the provision of **security checks** in early execution phases
- the ability to generate up-to-date **security configuration use cases** in the documentation.

This paper provides a motivating example in the form of Apache Brooklyn [8]. Brooklyn is one of the earliest automated configuration management tools that are now widely used by cloud engineers and designers. It is also one of the most flexible in that not only can be it used to configure cloud functions, storage, and infrastructure but it can also be used to orchestrate and deploy configurations to multi-cloud or hybrid cloud systems. Brooklyn clearly segregates configuration parameters (stored in YAML files) from the implementation and deployment scripts (which ultimately constitute a curated set of templates known as blueprints) that are recommended as starting points for new systems and applications. This

ICCBDC 2023, August 17–19, 2023, Manchester, United Kingdom

ACM ISBN 979-8-4007-0733-9/23/08.
https://doi.org/10.1145/3616131.3616142

separation was one of the motivating factors behind this research. It also led to additional important questions for this paper: how secure are the blueprints as starting points, and how susceptible are they to user misconfiguration errors?

The paper presents related work in Section 2 before presenting an overview of the solution in Section 3. The solution is evaluated against some real cloud application templates in Section 4 leading to discussion and final conclusion in Sections 5 and 6.

2 BACKGROUND

Cloud computing has evolved over the past twenty years from a model that fused together the cost benefits of virtualisation, the service-based flexibility of on-demand utility computing, and the always-on, high-availability network access of the internet to create a new paradigm [9]. As cloud services became more popular, both the cloud service providers (CSPs) and various third parties created tools to make designing and managing cloud applications easier, and to encourage new adopters. Apache Brooklyn, as well as being one of the earliest configuration automation tools also offers a combination of features not found in the other common configuration managers such as Chef and Puppet(ref). It is particularly suited as the focus of this research because of how it clearly segregates the configuration from the implementation and deployment stages. It also utilises standardised YAML-based specification languages from the Organisation for the Advancement of Structured Information Standards (OASIS) [10]. There have been numerous studies concerning where security vulnerabilities are introduced into the system life cycle. The publication by [11] discusses the issues of security vulnerabilities in requirements management, while Lamport introduced Temporal Logic of Actions (TLA+) [12] which was a formal specification language that he recommended (and used) to ensure that systems were designed and implemented in a more formal manner. Lamport himself argues that making proofs easier to understand requires only structure and naming [13]. More recently, Behaviour Driven Development (BDD) [14] built on the concept of structure and naming by introducing Cucumber and Gherkin. These are a combination of unit-test software and specification language, respectively, used for requirements gathering and verification. Unlike TLA+, the specification language (Gherkin) is more English-like and allows agile idioms such as user stories to be used to form BDD scenarios which can be directly used in Cucumber as unit tests. In this way, acceptance tests become executable andpart of the automated testing suite. There are also automated tools available for checking code quality, and dynamic and static execution, not to mention security.

User misconfiguration errors (UMEs) are the latest in a long line of security vulnerabilities that need to be addressed. UMEs, according to [3] (using concepts from Amazon Web Services) can be categorized into 6 groups: IAM (identity and access management) misconfiguration; S3 bucket (i.e. storage) misconfiguration; Access key misuse; Cognito (access token) misconfiguration; Elastic Block Store (EBS - i.e. storage) encryption misconfiguration; and security group misconfiguration (i.e. not restricting traffic to open ports).

Although recent technologies provide engineers with infrastructure-as-code (IaC) security testing tools [15] that can detect these types of vulnerabilities in IaC templates, such as Terraform and Cloud Formation code, fixing such errors in the implementation and/or deployment phases of the system development lifecycle is not ideal for a number of reasons. Firstly, research by [16] reported that "requirements and architecture design defects make up approximately 70% of all defects" but "80% of these defects are discovered late in the development life cycle" with a commensurate increase in software costs required to fix said errors in the later phases [16]. Secondly, a study published by Unit 42, an arm of Palo Alto Networks [17] found almost 200,000 IAC templates with the types of errors described herein, validating the research by [16] and demonstrating how easily such errors can be missed. Thirdly, fixing these errors in the later phases of the development lifecycle may lead to the fixes being subsequently overwritten if unpatched designs and/or source code are redeployed. Finally, fixing such errors using late-phase automatic checking tools misses an opportunity to improve developer knowledge, education, and documentation by failing to point out the errors they made in their initial configurations, as the research by [7] reported.

2.1 Motivation

There have been a number of papers published recently that highlight UMEs, As already discussed, [7] argue not just for testing tools but also earlier and more proactive checking tools, as well as for tools that can update UME use-cases in the system's documentation. Zhang et al. [18] discuss the issues of UME's in industrial control systems, and present a tool (SMFCONF) to address them. Guffey and Li [3], as discussed previously, present a study of the types of UMEs commonly encountered in cloud systems. This recent tranche of papers reflects the growing acknowledgment of the importance of, and risks associated with, UMEs.

2.2 Limitations of Existing Work

As previously mentioned, there are a number of commercial tools that check IaC templates for errors [15], including but not exclusively focusing on security. Recently, research by [19] brought together a number of tools to specifically check various IaC templates for security errors. The additional contribution of this paper is to perform similar checks, concentrating on the UMEs enumerated by [3], but performing these checks on the configuration file rather than the full IaC template itself. Our research shows that this "shift left" approach reduces the scan times by an order of magnitude and still detects the same errors.

3 OVERVIEW OF PROPOSED SOLUTION

In recent years, configuration and orchestration tools such as Apache Brooklyn [8] have added an element of modeling to the conceptualization phase, supplemented by specification modeling languages from OASIS, such as CAMP and TOSCA [20]. However, as discussed above, the (unintended) consequence of the facilitation of cloud system design is an increase in cloud security vulnerabilities attributable to UMEs, for the most part [21]. This work investigates the use of automated tools to help detect and ameliorate the effects of such UMEs while also attempting to improve the security knowledge and education of users. It does so by building a system termed CACLE - a knowledge-based security verification

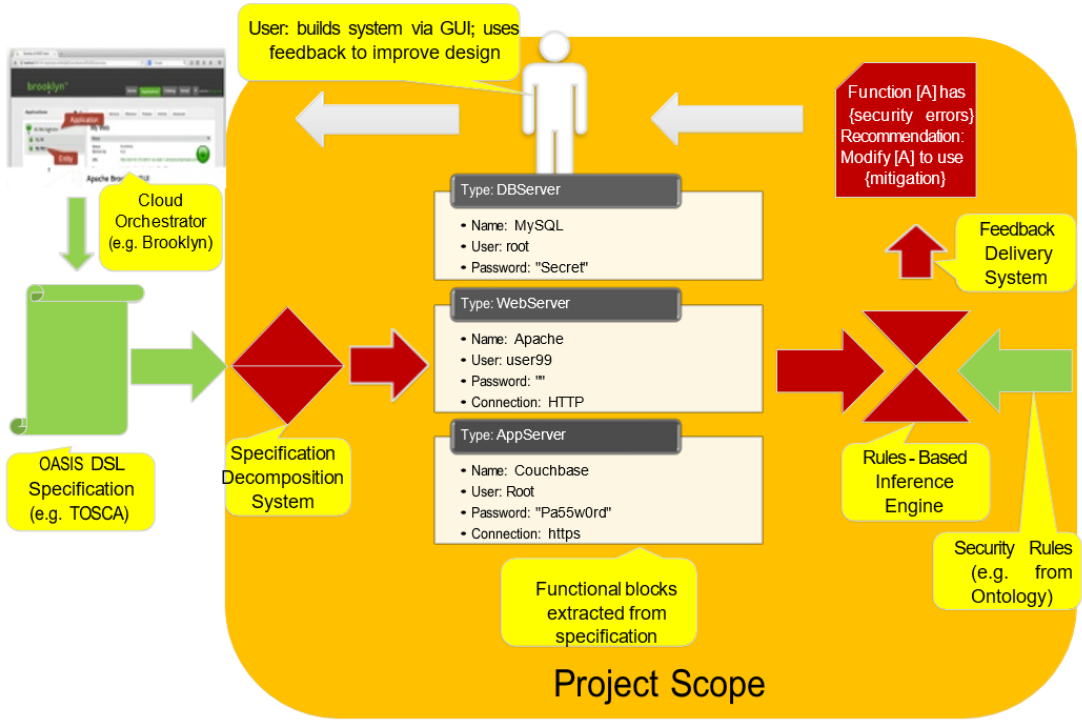


Figure 1: CACLE System Schematic showing Decomposer, Inferencing Engine, and Compliance Report

system, using security principles implemented as logic rules applied as part of an inference engine.

3.1 Modelling and Specification Languages

With regard to cloud environment configurations, the current standard offerings from OASIS are CAMP and TOSCA [8]. CAMP is a domain-specific language used to deploy application configurations to multi-cloud platforms, in particular, hybrid clouds, and was the forerunner of TOSCA. According to [8], TOSCA, along with CAMP, was designed to "define methodologies to describe and wrap the structure of cloud applications (components and relationships)". However, although the TOSCA standard, in particular, has been posited as a potential solution to the hybrid cloud challenge [22], the lack of homogeneous implementations between the various vendors has meant that a suite of tools compatible with **all** template versions is difficult to create. As other studies have shown [19], allowing for the additional processing overhead required to check the functional blocks and scripts required for a full deployment tends to increase the security checking time by an order of magnitude. Therefore, this paper proposes that rather than attempt to reason over the whole application configuration, we concentrate on extracting the essential security parameters for each component within the design - an approach based on research around composable systems [23]. This research has a wider relevance because TOSCA templates have also been used to create specifications for unikernels [24] and Internet of Things (IoT) systems [25], and it is suggested that this research may likewise be utilized in those domains as well.

3.2 System Description

The verification system starts with a script written in Python that extracts the pertinent security parameters from functional blocks within the YAML specification scripts. This approach was based on functional decomposition. These parameter sets are then passed on to the inference engine, which checks the parameters for security errors using a set of logic clauses. A schematic of the system is shown in Figure 1.

3.3 Graphs and Algorithms

The TOSCA and CAMP specification and modeling languages are good examples of a standard that can be used to provide platform-agnostic designs. However, they tend to have too much 'syntactic sugar' [26], especially from the perspective of security verification. To perform a security evaluation of a cloud specification, the system under consideration requires only the details of the functional blocks that will form the constituent parts of the proposed system. This solution was designed to parse the configuration files and decompose them into their constituent elements. The components of each element were then reviewed by the rules-based inference engine which made decisions based on a set of security rules, which in turn generated a report that enumerated the existing security vulnerabilities found in the templates and recommended potential mitigations. A graph representing the data flows is given in Figure 2.

The security rules utilized as part of CACLE are represented in an easily comprehensible, executable format that allows inferences to be made based on a security policy. These were based on previous

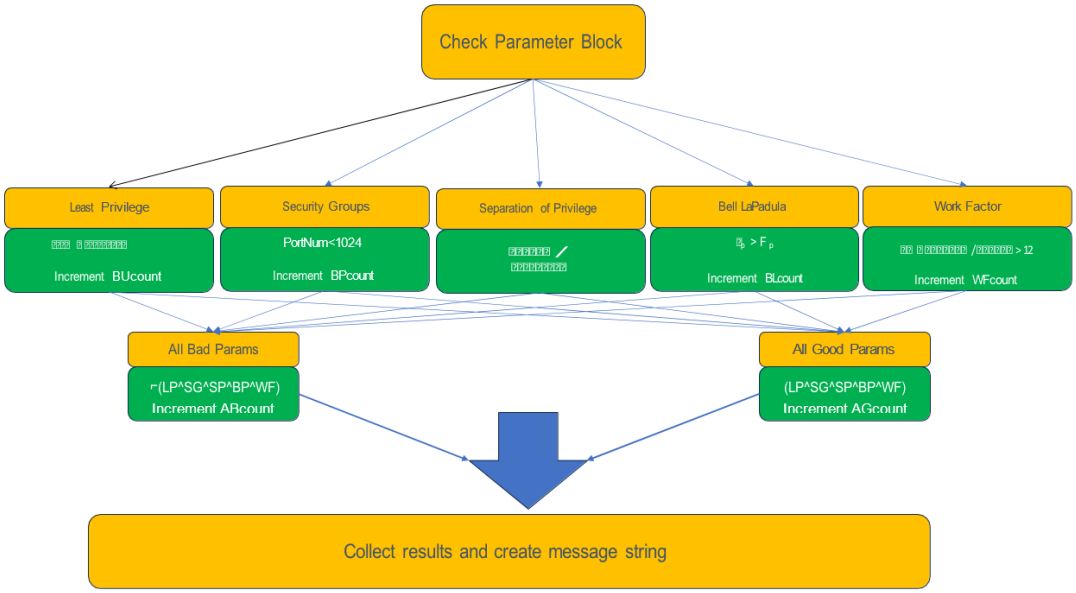


Figure 2: CACLE System Logic Graph

Table 1: Security Principles and Clauses

Principle	Logic Clause
Least Privilege	User(X) $\notin$ (SuperUser)
Separation of Privilege	GoodPW(X) $\wedge$ Goodtoken(Y)
Work Factor	(X $\notin$ (BadList) $\wedge$ $Length(X) \geq N$)
Bell LaPadula	Check (*UP, FP*)

research (eg. [27]) that posited the use of first-order logic statements (FOL) as a suitable mechanism in which to implement security rules as part of a knowledge-based inferencing system. To demonstrate the principle, this project implemented a subset of the rules detailed in Table 1, namely least privilege and work factor. These clauses were implemented using Maude rewriting logic rules, as a more modern and flexible alternative to more traditional formats such as Prolog and Datalog.

3.4 Implementation

The specification decomposition application was implemented in Python, based on an algorithm by Birchard [28]. It extracted the functional blocks and associated security attributes and passed them on to the rules engine. The inference engine was implemented in Maude, consisting of a set of equational and rewrite logic rules representing the security principles discussed previously. The recommendations of the inference engine represented as a vector string of characters, were passed back to a simple web service implemented using the Python flask web application framework [29].

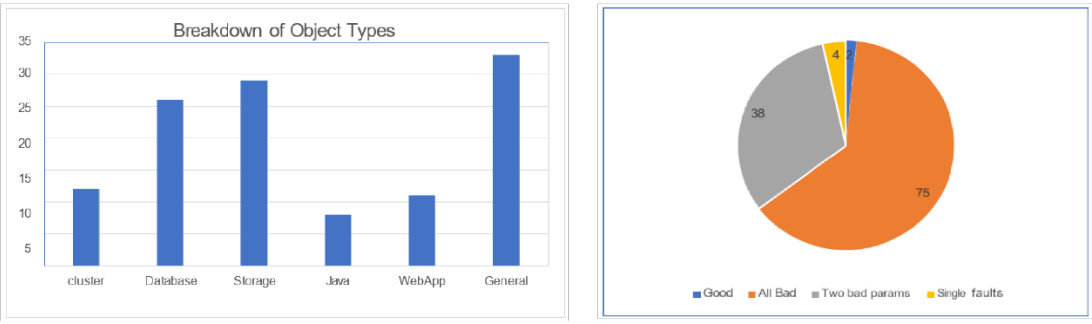
4 EVALUATION

4.1 Experimental Setup

Over one hundred candidate TOSCA and CAMP YAML files were gathered from various GIT repositories including Alien4Cloud, OASIS-Open, and Apache Brooklyn. All tests were carried out on a MacBook Air M2 2022 with 16GB of RAM, using Python version 3.9.6 and Maude version 3.1.

4.2 Evaluation methodology

As discussed previously, files were gathered from publicly available GitHub repositories. The initial objective was to utilize files from multiple vendors. However, despite the fact that TOSCA promotes itself as a standard for hybrid, multi-cloud, and even IOT system designs, the reality was that the different templates and structures used by different vendors made a single decomposition system unfeasible. Indeed, there is not even a parser that can verify all the various templates [30]. Ultimately, this evaluation settled on using files that adhered to the older CAMP specification, produced as 'blueprints' by the Apache Brooklyn project [8], as discussed previously. Due to their nature, these files exhibited greater uniformity and provided a repeatable structure that the decomposition



(a) Types of application specification files included

(b) Number of Parameter Sets with UMEs detected

Figure 3: Analysis of the Cloud Applications and Parameter Sets

Table 2: Comparison of Security Misconfiguration Errors

Category	Unit 42 [17]	CACLE
Weak Passwords	53 %	64.5 %
Exposed Ports	76 %	85 %
Excessive Permissions	99 %	98 %

system utilized to extract the essential security information from the template functional blocks.

As discussed above, the inference engine tested for least privilege (i.e. functions should not be executed under the administrator or superuser role), fail-safe defaults (i.e. any open ports should be specifically selected as well as not utilizing system port addresses), and work factor (i.e. a computational function derived from the complexity of the passwords selected). A comparison of how these tests mapped onto the UME categories from [3] is given in a later section. The blueprints included a varied selection of application specifications including database servers, storage appliances and collections (including buckets and cloud storage), Java application and Web application servers, general infrastructure components (such as routers and virtual network devices), and clustered instances of application servers. This gave a wide selection of devices and applications that are often the subject of UMEs. The selection of applications is shown in the bar chart, (Figure 3a). Subsequently, the files were searched looking for the 'services' keyword, and then the name (id), username('adminUsername'), password('adminPassword'), and port number('port') fields and associated data were extracted. These were validated individually against the security rules and any non-conformance or data omissions were recorded. The recommendations from the inferencing engine were then manually compared with the gathered templates to verify that all the in-scope UMEs were successfully identified.

4.3 Results

The rules engine tested each parameter field for transgressions against the stated rules with results shown in Table 2. Of the 124 files that were processed, 119 were successfully decomposed, and 5 exhibited extraction errors which meant that they could not be processed further. Of the testable files, the rules engine examined 354 parameters and found that 98% of the sets tested exhibited some security faults or errors (the 2 percent without faults were sets added by the researchers to confirm sets with no UMEs were successfully detected). A breakdown of the number of parameter sets with different errors is shown in Figure 3b. *Allbad* denotes the number of sets where all parameters were misconfigured; *Two bad* denotes parameter sets with two misconfigured parameters, *single faults* is the number of sets with only one misconfigured parameter and *Good* shows the number of parameter sets with no misconfigured parameters. Individual error counts are summarised in Table 2. These results were compared to data published by Unit 42, the security/ consulting group of Palo Alto [17]. The results were as displayed in Table 2 and an explanation of the results is given below.

- Password Strength: This research tested that passwords were not in an excluded/ blocked list and that the length was greater than 12 characters. Unit 42 only tested for a password length of 14 characters or more but did not employ a blocked list (This is equivalent to the IAM UME from [3]).
- Exposed Ports: This research tested for any exposure of system ports (i.e. numerically less than 1024) as being a potential security risk. Unit 42 only tested for SSH and RDP (This is equivalent to the security group UME from [3]).
- Excessive Permissions: Unit 42 tested for specific cloud system roles. CAMP is not an infrastructure-specific specification language and furthermore is vendor agnostic. Therefore, this research tested for names in an excluded/blocked list OR where default names were used/assumed OR where passwords were missing or easily guessable. (This effectively amalgamates the IAM and storage categories from [3]).
- From a performance perspective, in performing the above tests, the Maude system executed checks on data from 119

files in 58ms. The research by [19] tested 130 files in 5.13s (using their compatibility matrix) and 6.69s (without the matrix). This is more than an order of magnitude slower than the tests carried out by this team - although the former research did test a wider range of file types.

As the results show, virtually all of the sets tested had one or more basic security errors. These were successfully detected by the rules engine and flagged (with mitigations) to the user.

5 DISCUSSION

The initial premise of this research was to investigate whether testing the security of system design at the conceptual- ization phase of the system development life cycle could help reduce the number of UMEs found in modern systems, which ultimately lead to significant security errors. From that perspective, and in comparison with the results published by [17], the results suggest that testing of the configuration phase detects the same degree of security issues as testing at later phases. The differences in the results have been discussed above and can be attributed to variations in the test parameters and the scope of the design encapsulated by the specification template. TOSCA, ultimately, did not live up to the promise of its definition which, as with many other standards created by committees, was poorly supported by actual users of the templates. The result of this was that this research used an older standard as the subject of its tests. More importantly, the lack of standardization in TOSCA templates means that a much more sophisticated - and complex - decomposition application would be required to do similar research.

Another limitation encountered during this study was the Brooklyn blueprint templates. While these files may well be suitable for inexperienced users from a technical/infrastructure perspective, this research clearly shows that they do not meet even the most basic security requirements. Therefore, it is the authors' opinion that this is a serious omission because the typical user of such templates is often the same inexperienced cloud designer that is the subject of this research. Many blocks did not specify a username or a password. Even when they did specify a name, it was usually the superuser - a selection that countermands the most basic security tenet of least privilege. Additionally, even when a password was specified, it was either in the blocked list of most common passwords [31] or it was too short to meet modern accepted standards (such as those utilized by Unit 42 [17]). In fact, the only passwords that passed the test were the ones included to verify the system by the researchers. Finally, the last test concerned exposed ports. This research tested for any exposure of system ports because they both expose the services available and also provide a route for privilege escalation via published exploits - this is in the same vein as the security group UME discussed by [3]. Ultimately, the two sub-questions raised by the research, namely how secure are blueprints as starting points, and how susceptible are they to user misconfiguration errors, have been addressed by the results above. The default position of the templates is insecure, and because there is no checking or filtering between the configuration files and the blueprints used to create cloud (or IOT) systems, the eventual state is insecure as well. This would indicate a need for a checking solution (such as CACLE) to be interposed between the source files and the deployed files. This would be relatively easy to accomplish in the Brooklyn environment.

Another major omission of both the TOSCA and CAMP templates was the lack of support for roles and access control. This constituted another limitation of the proposed 'standard' because, as this research reveals, the limited amount of security information contained in the current templates severely restricts the amount and type of security validation tests that can be carried out on these files. Some work around adding additional security features to TOSCA has already been carried out by [24].

For the final stage - delivering messages to the user - the Maude-based inference engine assessed the security posture of the design specification and encoded it into a character representation. This resulting tag pattern was passed to a Python-based system that provided the user interface and also interpreted the results, producing feedback message(s) that could be read and acted on by the designer. This delivery system leverages the SWIG-based Python-Maude language bindings [32] to transfer the recommendations created by logic in Maude over to Python. In this way, the solution makes use of the best of both systems: using Maude to process logic and perform the inference, and using Python to store and process the text (and take advantage of libraries such as Babel for localization and internationalization), as well as implementing the delivery system to the user via built-in web components such as Flask. Further research is recommended, possibly using groups of users, to identify the best way of expressing and delivering the recommendations.

6 CONCLUSION

As discussed in the introduction, this research was designed to achieve three objectives:

- the provision of proactive parameter value checking and validation - focusing specifically on **security**
- the provision of **security checks** in early execution phases
- the ability to generate up-to-date **security configuration use cases** in the documentation.

The findings (see Table 2) indicate that validating at the conceptual design stage finds a very similar percentage of security defects to solutions that detect faults at a much later phase of the system development life cycle (such as those described by [15] or [17]). The results also show that the approach developed by the researchers has the potential to check for errors and produce recommendations much more quickly than existing solutions. Additionally, this could be further accelerated, if required, by the use of parallel processing due to the fact that parameters are checked independently and the recommendation section is the only place the results need to be combined and aligned. As discussed previously, the limitations of the TOSCA 'standard' (and in particular the lack of security parameters) restrict the number and type of tests that can be performed at the moment.

Additionally, the use of logic statements to represent security rules means that the range of rules that could be included can be easily expanded and the system is easily extended by the use of standard import functions in the Maude System. It is beyond the scope of this paper to investigate how effective the recommendations produced were. That will be left to subsequent research.

REFERENCES

[1] G. Alvarenga. *What is Shift Left Security*. Jan. 2022. https://www.crowdstrike.com/cybersecurity-101/shift-left-security/.

[2] D. M. Berry. "Formal methods: The very idea - Some thoughts about why they work when they work". In: *Science of Computer Programming* 42.1 (Jan. 2002), pp. 11–27.

[3] J. Guffey and Y. Li. "Cloud Service Misconfigurations: Emerging Threats, Enterprise Data Breaches and Solutions". In: *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, Mar. 2023, pp. 0806–0812.: https://ieeexplore.ieee.org/document/ 10099296/.

[4] M. Guerriero *et al.* "Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry". In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Sept. 2019, pp. 580–589. : https://ieeexplore.ieee.org/document/8919181/.

[5] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams. "A systematic mapping study of infrastructure as code research". In: *Information and Software Technology* 108 (Apr. 2019), pp. 65–77.: https://linkinghub.elsevier.com/retrieve/pii/S0950584918302507.

[6] C. A. Cois, J. Yankel, and A. Connell. "Modern DevOps: Optimizing software development through effective system interactions". In: *2014 IEEE International Professional Communication Conference (IPCC)*. IEEE, Oct. 2014, pp. 1–7.: https://ieeexplore.ieee.org/document/7020388/.

[7] Y. Zhang *et al.* "An Evolutionary Study of Configuration Design and Implementation in Cloud Systems". In: (Feb. 2021). http://arxiv.org/abs/ 2102.07052.

[8] J. Carrasco *et al.* "Bidimensional Cross-Cloud Management with TOSCA and Brooklyn". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, June 2016, pp. 951–955. http://ieeexplore.ieee.org/document/7820380/.

[9] B. Varghese. *History of the cloud*. Mar. 2019.: https://www.bcs.org/articles-opinion-and-research/history-of-the-cloud/.

[10] A. Brogi, A. Di Tommaso, and J. Soldani. "Validating TOSCA application topologies". In: *MODELSWARD 2017 - Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*. Vol. 2017-January. SciTePress, 2017, pp. 667–678.

[11] I. Kwan and D. M. Berry. *Specify First or Build First? Empirical Studies of Requirements Engineering Activities: A Survey*. Tech. rep. Waterloo: University of Waterloo, 2009.

[12] L. Lamport. "Real-Time Model Checking is Really Simple". In: *Microsoft Research* (2005).

[13] L. Lamport. "How to Write a 21 st Century Proof". In: ().

[14] C. Solis, X. Wang, and C. Solís. A study of the characteristics of behaviour driven development A study of the characteristics of behaviour driven development A Study of the Characteristics of Behaviour Driven Development. Tech. rep. 2011, pp. 383–387.: https://hdl.handle.net/10344/1256.

[15] D. P. Acharya. *5 Tools to Scan Infrastructure as Code for Vulnerabilities*. 2023.: https://geekflare.com/iac-security-scanner/.

[16] P. Feiler *et al.* "Four Pillars for Improving the Quality of Safety-Critical Software-Reliant Systems". In: *SEI-CMU* (2013), pp. 1–17. : www.sei. cmu.edu.

[17] M. Chiodi. *Cloud Threat Report: Putting the Sec in DevOps*. Tech. rep. Santa Clara: Unit 42, Apr. 2020.

[18] Q. Zhang *et al.* "Automated Runtime Mitigation for Misconfiguration Vulnerabilities in Industrial Control Systems". In: *25th International Symposium on Research in Attacks, Intrusions and Defenses*. New York, NY, USA: ACM, Oct. 2022, pp. 333–349. : https://dl.acm.org/doi/10.1145/ 3545948.3545954.

[19] N. Petrovic, M. Cankar, and A. Luzar. "Automated Approach to IaC Code Inspection Using Python-Based DevSecOps Tool". In: *2022 30th Telecommunications Forum, TELFOR 2022 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., 2022.

[20] A. Brogi, J. Soldani, and P. W. Wang. "TOSCA in a nutshell: Promises and perspectives". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8745 LNCS. Springer Verlag, 2014, pp. 171–186.

[21] U. D. Ani, H. He, and A. Tiwari. "Vulnerability-Based Impact Criticality Estimation for Industrial Control Systems". In: *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, June 2020, pp. 1–8. : https://ieeexplore.ieee.org/document/9138886/.

[22] M. Bowker and B. Laliberte. *Strategies for optimizing on-premises and public cloud infrastructure*. ESG, 2020.

[23] V. Roussev, P. Dewan, and V. Jain. "Composable collaboration infrastructures based on programming patterns". In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. New York, NY, USA: ACM, Dec. 2000, pp. 117–126. : https://dl.acm.org/doi/10.1145/ 358916.358982.

[24] M. Compastié *et al.* "A TOSCA-Oriented Software-Defined Security Approach for Unikernel-Based Protected Clouds". In: *IEEE Conference on Network Softwarisation*. 2019, pp. 151–159. https://hal.archives-ouvertes.fr/hal-02271520.

[25] S. Choudhuri. "A Case for Unikernels in IoT: Enhancing Security and Performance". In: *Internet of Things: Enabling Technologies, Security and Social Implications*. Springer, Singapore, 2021, pp. 85–91. http://link.springer.com/10.1007/978-981-15-8621-7_7.

[26] P. J. Landin. "The Mechanical Evaluation of Expressions". In: *The Computer Journal* 6.4 (Jan. 1964), pp. 308–320. https://academic.oup.com/ comjnl/article-lookup/doi/10.1093/comjnl/6.4.308.

[27] J. Mclean. *Security Models*. Tech. rep. 1994, pp. 1136–1145. http://citeseerx.ist.psu.edu/viewdoc/download?doi$=$10.1.1.34.8561&rep$=$rep1&type$=$pdf.

[28] T. Birchard. *Extract Nested Data From Complex JSON*. Oct. 2018. https://hackersandslackers.com/extract-data-from-complex-json-python/.

[29] M. R. Mufid *et al.* "Design an MVC Model using Python for Flask Framework Development". In: *2019 International Electronics Symposium (IES)*. IEEE, Sept. 2019, pp. 214–219. https://ieeexplore.ieee.org/document/8901656/.

[30] J. DesLauriers *et al.* "Cloud apps to-go: Cloud portability with TOSCA and MiCADO". In: *Concurrency and Computation: Practice and Experience*

[31] 33.19 (Oct. 2021). https://onlinelibrary.wiley.com/doi/10.1002/cpe.6093.

[32] Anon. *Top 200 Most Common Password List*. 2022. https://nordpass.com/most-common-passwords-list/.

[33] R. Rubio. "Maude as a Library: An Efficient All-Purpose Programming Interface". In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 13252 LNCS (2022), pp. 274–294. https://link.springer.com/chapter/10.1007/ 978-3-031-12441-9_14.