Verification of Cloud Security Policies

Loïc Miller
ICube
University of Strasbourg
Strasbourg, France
loicmiller@unistra.fr

Pascal Mérindol
ICube
University of Strasbourg
Strasbourg, France
merindol@unistra.fr

Antoine Gallais
ICube
UPHF / INSA Hauts-de-France
Valenciennes, France
antoine.gallais@uphf.fr

Cristel Pelsser
ICube
University of Strasbourg
Strasbourg, France
pelsser@unistra.fr

***Abstract*—Companies like Netflix increasingly use the cloud to deploy their business processes. Those processes often involve partnerships with other companies, and can be modeled as workflows where the owner of the data at risk interacts with contractors to realize a sequence of tasks on the data to be secured.**

In practice, access control is an essential building block to deploy these secured workflows. This component is generally managed by administrators using high-level policies meant to represent the requirements and restrictions put on the workflow. Handling access control with a high-level scheme comes with the benefit of separating the problem of specification, i.e. defining the desired behavior of the system, from the problem of implementation, i.e. enforcing this desired behavior. However, translating such high-level policies into a deployed implementation can be error-prone.

Even though semi-automatic and automatic tools have been proposed to assist this translation, policy verification remains highly challenging in practice. In this paper, our aim is to define and propose structures assisting the checking and correction of potential errors introduced on the ground due to a faulty translation or corrupted deployments. In particular, we investigate structures with formal foundations able to naturally model policies. Metagraphs, a generalized graph theoretic structure, fulfill those requirements: their usage enables to compare high-level policies to their implementation. In practice, we consider Rego, a language used by companies like Netflix and Plex for their release process, as a valuable representative of most common policy languages. We propose a suite of tools transforming and checking policies as metagraphs, and use them in a global framework to show how policy verification can be achieved with such structures. Finally, we evaluate the performance of our verification method.

***Index Terms*—policy verification, metagraphs, policy modeling, rego, access control, authorization**

I. Introduction

Authorization is a key aspect of security, regulating the interactions taking place in a given system. For example, Netflix will often interact with their partners for some tasks, e.g. content ingestion [1], [2]. Since the systems to be secured by authorization can be highly complex, administrators often rely on policy-based management of authorization. Policies define the desired behavior of a system from a high-level perspective. Hence, this form of management allows to separate the problem of specification, i.e. defining the desired system behavior, from the problem of implementation, that is the enforcement of the desired system behavior.

Research on this topic mainly focuses on three areas: policy analysis, policy refinement and policy verification. On the one hand, **policy analysis** deals with the fulfillment of specific properties by a set of policies [3], e.g. detecting when two or more policies are conflicting. On the other hand, **policy refinement** handles the translation from high-level policies into low-level configurations [4]. Depending on how this task is realized, the translation can lead to incorrect and/or non-optimized policy implementations; it can affect performance or even put the system at risk by introducing security flaws. According to the Verizon Data Breach Investigations Report, errors were causal events in 22% of data breaches [5]. As the risk of error increases when refinement is performed by hand, automatic or semi-automatic assisting tools have emerged to help administrators better translate their policies [6]–[8].

In this paper, we deal with **policy verification**, i.e., we check whether the deployment of policies actually meets their high-level specification. Policy verification plays an important role since assisting tools are not free of errors, and deployment specificities can lead the policy to become erroneous. An erroneous policy can lead attackers to view files they were not authorized to see [9], access paid content free of charge [10] and even changing access rights [11] or deleting content [12]. There exists only few works on policy verification [13], [14], when compared to the large body of work dealing with policy analysis, and none of them uses metagraphs. We propose to model policies with a generic yet rich structure: metagraphs. We use its formal foundations to verify whether the actual deployment of a policy (i.e., its implementation) matches its initial specification. We rely on this structure since, by design, it provides means to locate conflicts and avoid redundancy [15]. Metagraphs provide more fine-grained verification process than with other structures like usual graphs. To the best of our knowledge, metagraphs belong

This project has been made possible in part by a grant from the Cisco University Research Program Fund, an advised fund of Silicon Valley Foundation.

978-1-6654-4005-9/21/$31.00 ©2021 European Union

to the rare appropriate structures able to naturally model access control policies. Metagraphs also scale well with very large policies, since we use their formal foundations to perform the verification. Transformations such projections or context metagraphs [16] can be used to help with the visualization of very large policies.

The contributions of our paper are as follows:

1) We are the first to use metagraphs to perform the verification of access control policies. We argue they represent one of the most appropriate form of policy modeling enabling refinement and verification. We also show how this verification allows us to pinpoint errors in the policy.
2) We propose a suite of translation tools enabling policy verification. More specifically, we introduce how to perform policy verification on a workflow-like policy specification. We rely on a policy implementation based on Rego, a high-level declarative language built for expressing complex policies.
3) Finally, we conduct a thorough performance evaluation. We verify that deployed policies match their specification in a very reasonable time, even for large workflows having a substantial number of rules.

II. Related Works

There exist several pieces of works on policy analysis, refinement and verification in the literature. Policy analysis mainly deals with policy evaluation and anomaly analysis: checking for errors like incorrect policy specifications, conflicts and sub-optimizations affecting either a single policy or a set of policies [3] being the primary research topic. Works in this area use different techniques to achieve this goal, such as model checking [17], [18], binary decision diagrams [19], graph theory [20], Deterministic Finite State Automata (DFSA) [21], First Order Logic (FOL) [22], geometrical models [23], answer set programming [24], petri nets [25] and metagraphs [15]. Policy evaluation instead deals with checking whether a request is satisfied by a set of policies. It is typically used to verify the effective impact of modifying a policy. Works that deal with analyzing the impact of changes in a policy usually model those policies and then analyze the obtained representation for effective impact [26], [27].

Policy verification, the subject of this paper, deals with checking whether a policy is correctly enforced in a system. On the one hand, Hughes and Bultan [13] as well as Bera et al. [14] propose automatic verification of access control policies against a set of properties. Verification is achieved by translating the properties into a boolean satisfiability problem and using a SAT solver, whereas we use metagraphs which come with a useful visual representation of the policies. On the other hand, even though metagraphs have emerged as one of the most suited tool for representing and reasoning about policies, they are still underused with only few existing works in the literature [15], [16], [28]–[30]. Basu and Blanning [16] compiled all the research on metagraphs up until 2007 in a book, which is the reference for general metagraph theory and applications. Ranathunga et al. [28] defined a toolkit in python to manipulate metagraphs. Hamza et al. [29], [30] used metagraphs to model policies in IoT devices to generate and validate Manufacture Usage Descriptions (MUD) profiles – it can be used to define the access control model and network functionality these devices require to properly function. They also check compliance of those MUD profiles with different levels of security policies, to determine where those devices are safe to be deployed. They do not verify policy implementations against their specifications. Closer to our contribution, Ranathunga et al. [15] use metagraphs to model network policies for distributed firewalls. In particular, they use specific metagraph properties to detect redundancies and conflicts in those policies. Contrary to our work, they do not verify deployed policies against specifications.

After introducing metagraphs (Sec. III), we show how we use them to perform policy verification (Sec. IV). We evaluate our approach (Sec. V), and finally conclude (Sec. VI).

III. From Graphs to MetaGraphs

A metagraph is a generalized graph theoretic structure like directed hypergraphs, which is defined as a collection of directed set-to-set mappings. Each set (containing subsets or elements) in the metagraph is a vertex, and directed edges represent the relationship between sets. More formally, a metagraph can be defined as follows:

Definition 1 (Metagraph)**.** *A metagraph $S = \langle X, E \rangle$ is a graphical construct specified by a generating set X and a set of edges E defined on the generating set. A generating set is a set of elements $X = \{x_1, x_2, ..., x_n\}$, which represent variables of interest. An edge e is a pair $e = \langle V_e, W_e \rangle \in E$ consisting of two sets, an invertex $V_e \subset X$ and an outvertex $W_e \subset X$.*

Fig. 1 illustrates a conditional metagraph. Conditional metagraphs are metagraphs augmented by propositions, i.e. statements that can either be true or false. A proposition attached to an edge must be true in order for the edge to be used in a path. Each edge may contain zero or more propositions and each proposition may be used in multiple edges. Overall, Fig. 1 represents the necessary tasks for employees to perform a bank transfer. Edges (e_1, e_2, e_3) relate sets of employees (u_1, u_2) and tasks ($create\_form$, $fill\_form$, $review\_form$, $transfer\_money$). They contain an arbitrary number of propositions, e.g. $tenure > 2$ for e_1. Using an edge depends on the evaluation of its propositions, e.g. both employees can perform the operations $create\_form$ and $fill\_form$ via e_1 provided they have more than two years of experience. In Fig. 2, we model a workflow containing temporal constraints as a conditional metagraph.

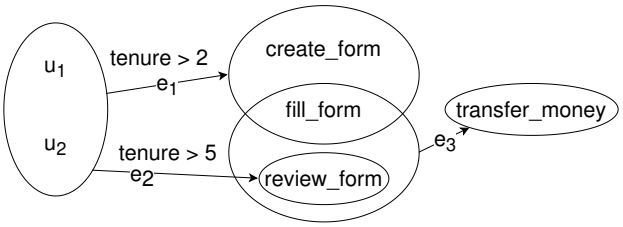

Fig. 1. A simple example of conditional metagraph to model the following question: what are the necessary tasks for employees to perform a bank transfer?

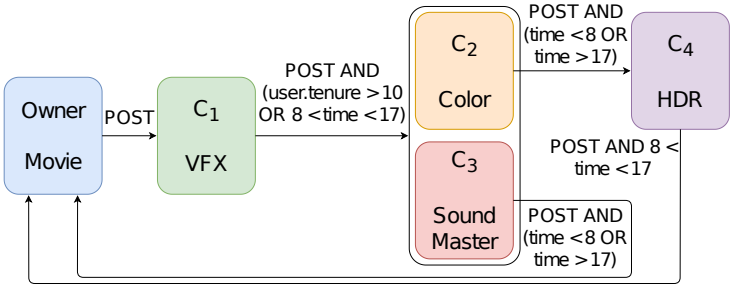

Fig. 2. Movie workflow: special effects apply before color tuning and sound mastering. HDR is set up last.

The notion of simple path, i.e. a sequence of edges $\langle e_1, e_2, ..., e_n \rangle$ from an element x to an element y where $x \in invertex(e_1)$, $y \in outvertex(e_n)$ and for all e_i, $i = 1, ..., n-1$, $outvertex(e_i) \cap invertex(e_{i+1}) \neq \varnothing$, does not describe all the connectivity properties existing in a metagraph. For example, in Fig. 1, there are two simple paths from $\{u_1, u_2\}$ to $\{transfer\_money\}$, (e_1, e_3) and (e_2, e_3). However, none of them can perform $transfer\_money$ as they respectively do not reach either $review\_form$ or $fill\_form$, which are both necessary to perform $transfer\_money$. Using the set consisting of all three edges (e_1, e_2, e_3) is necessary (and sufficient) to perform $transfer\_money$, but it is not a simple path: there does not exist a simple sequence of edges consisting of these three. Such a set of edges $\langle e_1, e_2, e_3 \rangle$ is called a metapath [16].

Reachability between the source and target sets in a metagraph is defined by the existence of (valid) metapaths between the two. Additionally, metapaths have a dominance property which can be used to determine redundant components (edges or elements) [15]. Once identified, those components can be safely removed from the policies. A metapath is *input-dominant* if no proper subset of its source is also a metapath to its target, *edge-dominant* if no proper subset of its edges is also a metapath to its target, and *dominant* if it is both input-dominant and edge-dominant [16].

IV. Policy Verification Using Metagraphs

By modeling the high-level policy specification as well as the translated policy implementation as two metagraphs, we can compare both in order to track (distributed) deployment errors. When specification and implementation metagraphs match, the policy implementation has been correctly translated from the policy specification. If they do not match, the metagraphs are not equivalent: errors occurred during the refinement and/or deployment. Fig. 3 summarizes our approach.

For our purposes and evaluations, we consider the verification of policies enforcing workflows. For instance, in the post-production stage of making a movie, the owner of the data at risk wants to employ other companies to edit its video and audio components; more specifically, the owner may want to add special effects (VFX), tune colors, set up High Dynamic Range (HDR) and master the audio. The intent of the owner can be modeled under the form of a workflow, as depicted in Fig. 2. Propositions on the edges constrain the communications. For example, C_1 can only send data to C_2 and C_3 if the communication is a `POST` request, and either the tenure of the user is greater than 10, or the request happens between 8 AM and 5 PM.

This form of policy specification can be generically expressed as a list of rules: each describing an edge of the metagraph, as a triplet of the form of $\langle$*source, destination, policy*$\rangle$. To implement those policies, we consider Rego, a high-level declarative language built for expressing complex policies. Once we have the policy specification and the implementation, we transform both into conditional metagraphs. For this, we develop three generic policy translators: from specification (raw) to specification metagraph, from specification to implementation (Rego), and from implementation to implementation metagraph.

Policy specification into a conditional metagraph – as denoted ② in Fig. 3: We need to define the variables set, the propositions set and the edge set defining the conditional metagraph. To this end, we parse the triplets of the policy specification file. A proposition attached to an edge must be true for the edge to be used in a metapath, thus, an `OR` in a proposition can be viewed as separate edges from the same source to the same destination, with each part of the `OR` becoming a sub-proposition attached to one of the newly created edges. Likewise, the `AND` in a proposition means both parts need to be true in order for the edge to be used, so the proposition can't be separated.

Policy implementation (i.e. Rego) into a conditional metagraph – ④ in Fig. 3: We use ANTLR4, Another Tool for Language Recognition, which is a parser generator used for translating structured files. After constructing our lexer rules and parser grammar for Rego, we were able to generate the Abstract Syntax Tree (AST) for any Rego policy file.

Comparing metagraphs – see ⑤: To compare metagraphs, we tag edges in one metagraph upon a match with edges in the other metagraph. Non tagged edges correspond to errors/mistakes in the implemented policies, singled out by our comparison.

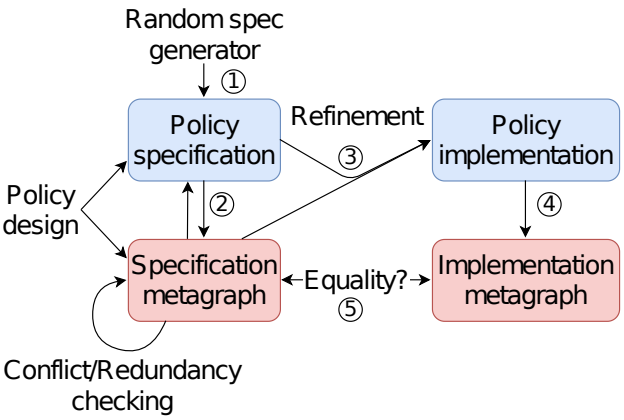

Fig. 3. Enabling policy verification using metagraphs. We propose 4 tools: ① RandomWorkflowSpecGenerator; ② ③ SpecToRego; ④ RegoToMetagraph; ⑤ SpecImplEquivalence.

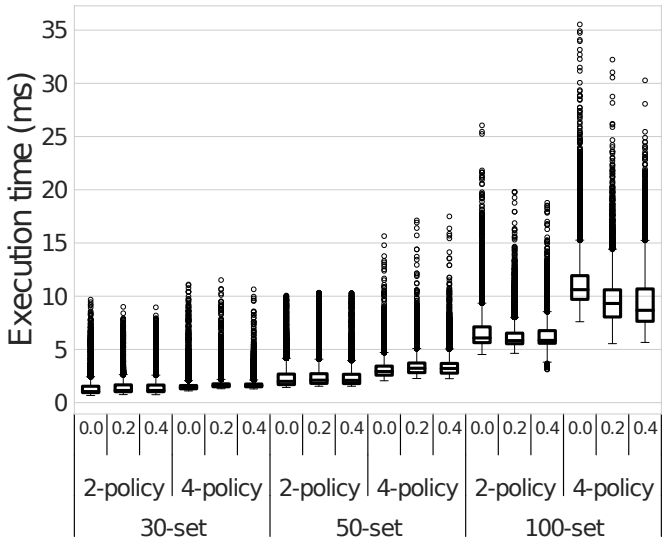

Fig. 4. Execution time of our matching algorithm according to several different parameters used to generate our access control policies. Values for 10-set and 20-set omitted for brevity.

V. Performance analysis

To evaluate our policy verification algorithm, we measure the time required to compare the specification and implementation metagraphs.

Methodology: To obtain generally representative results, we chose to generate random workflows for the access control policy specifications (instead of relying on few small real cases) based on the following varying parameters. Values for those parameters were chosen for the sizes of the generated policies to be in line with papers that model real-world policies [15], [31].

- **Size of the workflow**, i.e., number of elements in the generating set: 10, 20, 30, 50 or 100.
- **Policy size**, i.e., number of conditional propositions on each edge for the policy: 2 or 4.
- **Error rate**, i.e., fraction of errors in propositions of the metagraph. A value of 0.4 means that 40% of the elements/propositions of the metagraph are tampered with; we consider the following error rates: 0.0, 0.2 and 0.4.

Overall, we obtain 300 different policy specifications and 27,000 policy implementations (300 specifications, 3 error rates, 30 repetitions). After generating the policy implementations, we translate those into metagraphs to finally perform the comparison.

Evaluation: For each of the 27,000 scenarios, we measure the cumulative time of both sorting and matching for 30 runs, ending up with a total of 810,000 measures. We remove outliers due to peak machine load (Z-score superior to three): 9367 values out of 810000 (1.16%). We ran our measurements on commodity hardware with an Intel Core CPU 3.5-GHz, 16GB of RAM.

Fig. 4 shows that the error rate produces a negligible effect on the time required for the comparison, whereas the computing time increases with the number of elements in the generating set (as well as with the policy size). The main complexity of our metagraph comparison is inherent to edge sorting: $O(m{\cdot}log(m))$, with m the number of edges. Using an OLS regression, we confirm that the number of edges ($\beta = 0.0025$; $p < 0.001$) is a significant predictor. The overall model fit is: $R^2_{adj} = 0.868$, with the *post hoc power analysis* indicating a power greater than .999.

In summary, we argue that our refinement/deployment error detection technique can be efficiently implemented as long as the number of propositions of the policy is reasonable. The complete results and code are publicly available[1].

VI. Conclusion

Some of the largest cloud consumers use the cloud to deploy their workflows and enforce their processes using access control policies. In this paper, we detailed to what extent metagraphs are appropriate structures to naturally model access control policies. Their formal and graphical foundations guide the reasoning to manipulate such policies and provide means to detect conflicts and mitigate redundancies. This structure is a suitable modeling tool; while it enables policy analysis, we have proposed here to use them for a practical verification of the deployed access control policy regarding its specification. Our proposal compares the initial metagraph specification to its deployed counterpart and reveals inconsistencies. We evaluate the complexity of our simple but original approach to show its scalability.

[1]See https://zenodo.org/record/4426675.

References

[1] N. T. Blog. (2016) Netflix conductor: A microservices orchestrator. [Online]. Available: https://netflixtechblog.com/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40

[2] N. T. Blog. (2019) Evolution of netflix conductor: v2.0 and beyond. [Online]. Available: https://netflixtechblog.com/evolution-of-netflix-conductor-16600be36bca

[3] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2601–2614, 2017.

[4] J. D. Moffett and M. S. Sloman, "Policy hierarchies for distributed systems management," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 9, pp. 1404–1414, 1993.

[5] V. Enterprise. (2020) Data breach investigations report. [Online]. Available: https://enterprise.verizon.com/resources/reports/2020/2020-data-breach-investigations-report.pdf

[6] Amazon. (2020) Aws policy generator. [Online]. Available: https://awspolicygen.s3.amazonaws.com/policygen.html

[7] O. Dohndorf, J. Kruger, H. Krumm, C. Fiehe, A. Litvina, I. Luck, and F.-J. Stewing, "Tool-supported refinement of high-level requirements and constraints into low-level policies," in *2011 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 2011, pp. 97–104.

[8] K. Klinbua and W. Vatanawood, "Translating tosca into docker-compose yaml file using antlr," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2017, pp. 145–148.

[9] vulners. (2017) Razer us: Database credentials lea. [Online]. Available: \url{https://vulners.com/hackerone/H1:293470}

[10] C. Cimpanu. (2018) Steam bug could have given you access to all the cd keys of any game. [Online]. Available: https://www.zdnet.com/article/steam-bug-could-have-given-you-access-to-all-the-cd-keys-of-any-game/

[11] L. Muthiyah. (2018) Hacking facebook pages. [Online]. Available: https://thezerohack.com/hacking-facebook-pages

[12] A. Aboul-Ela. (2014) Delete credit cards from any twitter account. [Online]. Available: \url{https://hackerone.com/reports/27404}

[13] G. Hughes and T. Bultan, "Automated verification of access control policies using a sat solver," *International journal on software tools for technology transfer*, vol. 10, no. 6, pp. 503–520, 2008.

[14] P. Bera, S. K. Ghosh, and P. Dasgupta, "Policy based security analysis in enterprise networks: A formal approach," *IEEE Transactions on Network and Service Management*, vol. 7, no. 4, pp. 231–243, 2010.

[15] D. Ranathunga, M. Roughan, and H. Nguyen, "Verifiable policy-defined networking using metagraphs," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[16] A. Basu and R. W. Blanning, *Metagraphs and their applications*. Springer Science & Business Media, 2007, vol. 15.

[17] K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, and S. Chapin, "Automatic error finding in access-control policies," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 163–174.

[18] A. Khurat, B. Suntisrivaraporn, and D. Gollmann, "Privacy policies verification in composite services using owl," *Computers & Security*, vol. 67, pp. 122–141, 2017.

[19] H. Hu, G.-J. Ahn, and K. Kulkarni, "Discovery and resolution of anomalies in web access control policies," *IEEE transactions on dependable and secure computing*, vol. 10, no. 6, pp. 341–354, 2013.

[20] M. Koch, L. V. Mancini, and F. 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.

[21] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 1, pp. 30–50, 2000.

[22] M. Cheminod, L. Durante, F. Valenza, and A. Valenzano, "Toward attribute-based access control policy in industrial networked systems," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–9.

[23] C. Basile, D. Canavese, C. Pitscheider, A. Lioy, and F. Valenza, "Assessing network authorization policies via reachability analysis," *Computers & Electrical Engineering*, vol. 64, pp. 110–131, 2017.

[24] M. Rezvani, D. Rajaratnam, A. Ignjatovic, M. Pagnucco, and S. Jha, "Analyzing xacml policies using answer set programming," *International Journal of Information Security*, vol. 18, no. 4, pp. 465–479, 2019.

[25] H. B. Attia, L. Kahloul, S. Benhazrallah, and S. Bourekkache, "Using hierarchical timed coloured petri nets in the formal study of trbac security policies," *International Journal of Information Security*, vol. 19, no. 2, pp. 163–187, 2020.

[26] A. X. Liu, F. Chen, J. Hwang, and T. Xie, "Xengine: a fast and scalable xacml policy evaluation engine," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 265–276, 2008.

[27] A. X. Liu, F. Chen, J. Hwang, and T. Xie, "Designing fast and scalable xacml policy evaluation engines," *IEEE Transactions on Computers*, vol. 60, no. 12, pp. 1802–1817, 2010.

[28] D. Ranathunga, H. Nguyen, and M. Roughan, "Mgtoolkit: A python package for implementing metagraphs," *SoftwareX*, vol. 6, pp. 91–93, 2017.

[29] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman, "Clear as mud: generating, validating and applying iot behavioral profiles," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, 2018, pp. 8–14.

[30] A. Hamza, D. Ranathunga, H. H. Gharakheili, T. A. Benson, M. Roughan, and V. Sivaraman, "Verifying and monitoring iots network behavior using mud profiles," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[31] D. Ranathunga, M. Roughan, H. Nguyen, P. Kernick, and N. Falkner, "Case studies of scada firewall configurations and the implications for best practices," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 871–884, 2016.