An Authorization Service supporting Dynamic Access Control in Manufacturing Systems

Ivan Radonjić*, Enna Bašić*, Björn Leander*†, Tijana Marković*
*Mälardalen University,Västerås, Sweden,
{irc21001, ebc21002}@student.mdu.se, {bjorn.leander, tijana.markovic}@mdu.se
†ABB AB, Process Control Platform,Västerås, Sweden, bjorn.leander@se.abb.com

***Abstract*—Cybersecurity is of increasing importance in industrial automation systems. The use of fine-grained and intelligent access control is paramount in emerging manufacturing systems as implicit trust is no longer a viable assumption for interactions within industrial systems. An authorization service is a central component of an access control enforcement architecture, to which resource servers may outsource parts of the policy decision functionality.**

This paper investigates how to create and integrate an authorization service in an industrial manufacturing system, which uses workflow descriptions combined with operational system states for policy decisions. The implementation is demonstrated in the use case of recipe orchestration in a modular automation system, and a few key quality metrics of the authorization service are evaluated.

I. Introduction

Industrial Control Systems (ICSs) are essential for managing and supervising operations in industries such as manufacturing, energy production, water management, critical infrastructure, and transportation. These systems are enhancing industrial processes, improving efficiency, and minimizing human errors [1]. ICSs are transforming due to the evolution of Industry 4.0 [2], resulting in systems of increased connectivity and flexibility. Such change requires adjustments in the current cybersecurity measures, in charge of protecting computer systems, networks, and data from unauthorized access. Access control, as an essential security mechanism, is greatly impacted by this ongoing transformation. The traditional security models based on implicit trust are less useful in these emerging industrial systems, prompting a shift toward a zero-trust [3] approach. In a zero-trust architecture [4], any component interaction should be intelligently verified and evaluated, making detailed and dynamic access control a prerequisite.

Traditional static access control methods cannot adapt to changing operational conditions or the security status of the system [5]. In contrast, dynamic access control offers an enhanced approach for applying and managing access control, by establishing policies that are modified in real-time, based on the security environment [6] and operational conditions. Nevertheless, dynamic access control is still not widely utilized in ICSs.

Various methods of dynamic access control offer adaptable active permissions in response to system changes. Examples of such methods include following active workflows [7], [8] or using dynamic environmental conditions by implementing different methods of attribute-based access control [9], [10].

While manufacturing systems are becoming more dynamic and interconnected, the zero-trust model indicates that its access control mechanisms must exhibit similar dynamicity. To implement an access control enforcement architecture capable of providing sufficiently dynamic authorization, it is suggested that policy decisions are outsourced to an Authorization Service (AS) [11]. An AS can make central policy decisions based on running workflows or other environmental conditions. To the best of our knowledge, there is currently no mechanism that is able to synchronize an AS with the running workflows of a manufacturing system and transform that knowledge into access control policy decisions.

This paper presents an implementation of an AS which can make policy decisions in line with dynamically changing operational requirements, on behalf of resource servers in an industrial manufacturing system. The research builds on the idea of using knowledge of the workflow as a base for policy rule inference, as described by, e.g., Leander *et al.* [12]. The architecture and basic protocol used are adapted from [11]. The main contributions presented in this paper are:

- An implementation of an AS that can be integrated into a modular automation system.
- An implementation of a method that uses information about workflows as the base for access control policy decisions.
- An implementation of three different algorithms for encoding policy decisions into authorization tokens.
- An experimental evaluation of the implemented algorithms, concerning time and memory consumption and token size.

The remainder of this paper is organized as follows. Section II describes the related works. In Section III implementation details are provided, and Section IV contains an example of the use of the implemented solution. Section V presents the results, and in Section VI we discuss the impact and shortcomings. Section VII concludes the paper and outlines future plans.

II. Related Work

In a study conducted by Leander *et al.* [11], four access control enforcement architectures for dynamic manufacturing systems are suggested and evaluated. To alleviate complex

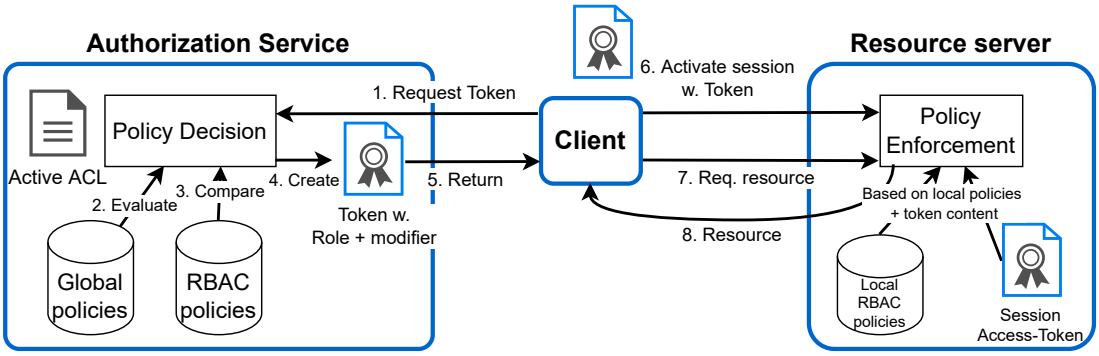

Fig. 1: Illustration of authorization protocol, from [11].

computational and communication burdens from the resource server, the usage of policy delegation mechanisms by use of ASs is advocated. Furthermore, some ideas on how to encode dynamic policy decisions in JSON Web Tokens (JWT) [13] are discussed. Our paper can be seen as a continuation of this work, as we use the same authorization protocol, illustrated in Fig. 1, but with a focus on the inner workings of the AS.

Other works are looking at access control enforcement architectures for industrial systems. For example, Martinelli *et al.* [14] and Watson *et al.* [15] both look at access control models and architectures in relation to the Open Process Communication Unified Architecture (OPC UA) [16], similar to this work. However, none of them looked at policy delegation mechanisms, which is the focus of this paper.

III. Design

This section presents the implementation details of the protocol for dynamic enforcement of access control, including the token population algorithm used to define explicit permissions and restrictions.

A. Authorization protocol

The protocol introduced in Section II is implemented to achieve the dynamic enforcement of access control. The implementation of the protocol is done in such a way that an AS monitors the active steps and workflows or recipes, and issues policy decisions based on that knowledge.

The assumption used in this work is that it is possible to separate the policy decision from the policy enforcement, so that an AS makes the policy decision on the request of a client, encodes that decision into an access token, and sends it back to the client so that the client can use the token for session activation towards a resource server. The resource server uses the token to make policy enforcement on resource requests from the client.

In Fig. 1, steps 1, 5, 6, 7 and 8, are part of the OPC UA indirect authorization protocol[1], while steps 2, 3, and 4 are related to the inner functionality of the AS, and are explained in the following subsections.

[1]reference.opcfoundation.org/GDS/v105/docs/9.6.5

B. Creating a per-session access control list based on global policies

In this work, the most restrictive access control strategy defined by [12] is implemented, i.e., "Entities assigned to a certain workflow are allowed to perform operations, following the sequence of the workflow." The permissions on the relationship between a client and a resource server are therefore limited by the active workflows in the system. Sequential Function Charts (SFC) [17] are used as the representation of workflows. An SFC contains a series of steps and transitions. Each step in the SFC contains zero or more operations, and each transition is guarded by a condition. Each operation and condition evaluation can be seen as a resource request toward a specific resource server, and can therefore be transformed into one or more permissions. An illustration of an SFC is represented in Fig. 2.

When the AS receives a token request from a client, the policy decision is based on knowledge of the active workflows and active workflow steps. Permissions are granted from active steps related to the resource server if the workflow for the step(s) is assigned to the client. The policy decision, represented by arrow 2 in Fig. 1, is in the form of a temporary Access Control List (ACL) and contains the set of granted permissions that the client will hold relative to the resources of the resource server for the duration of the session.

C. Token population

When the policy decision is taken, represented by the ACL in the previous section, the decision must be encoded into an authorization token, represented by arrow 4. in Fig. 1. This can be done directly using the explicit permissions in the ACL directly, or it may be needed to encode the policy decision in another form, depending on resource server capabilities.

It is a common characteristic that a resource server following the OPC UA standard has an internal representation of roles and permission[2], represented as local Role Based Access Control (RBAC) policies. This provides an opportunity to formulate the policy decision as a combination of roles, explicit permissions, and restrictions, which may be more

[2]UA Part 18, Role-Based Security (reference.opcfoundation.org/Core/Part18/v105/docs/)

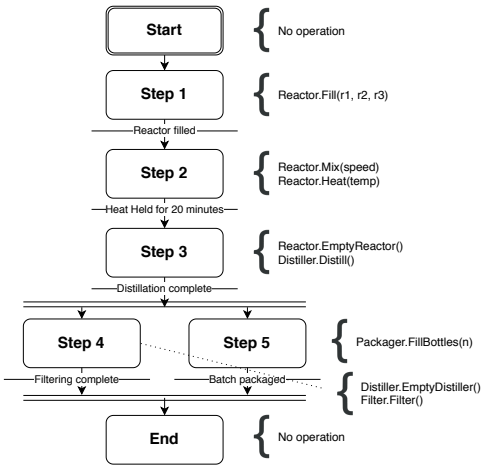

Fig. 2: Example of an SFC, from [12].

compact than the full ACL. To combine RBAC roles and explicit permissions and restrictions based on the active ACL, the algorithm suggested in [11] is implemented and adapted, represented by arrow 3. in Fig. 1.

The purpose of the token population algorithm is to get the minimum size token provided by the AS, which represents all of the permissions on the relationship between the client and the server. The concept relies on simplifying the token by identifying overlapping roles with extensive permissions, allowing the role to replace the long list of permissions.

Fig. 3 contains a flow chart illustrating the complete flow of the token population algorithm. Firstly, the algorithm initializes empty sets for the roles and restrictions. It creates a set of permissions P and populates it with permissions from the ACL. The algorithm searches the RBAC policies related to the resource server for the roles that have a set of permissions with the highest overlap with the P set. The computation of overlap involves counting the number of permissions that exist in both sets. If there is an overlap, the algorithm checks whether the addition of the role would make the total size of the token smaller. If so, the role is added to the roles set, and the overlapping permissions are subtracted from the permissions set P. Subsequently, any permissions granted by roles that are not in the ACL, are added to the explicit restrictions list. The process continues until no overlapping roles are found or there are no permissions left in the permissions set. In the case that there are no roles that have overlapping permissions with the ACL permissions, or if the size of the token has not decreased, all of the permissions from ACL are encoded into the token as explicit permissions. After populating the roles, permissions, and restrictions sets, they are encoded as claims in the final token, which is then signed, ensuring its authenticity and integrity.

Three approaches of the token population are implemented:

1) **Baseline approach** - An initial variation of encoding policy decisions into an authorization token, is to directly use the ACL permissions, without attempting any optimization, i.e., leaving the ”restrictions” and ”roles” claims empty. Because this approach does not employ the token population algorithm or any optimization, it is ideal as the baseline for comparing different encoding algorithms.
2) **Token Population Algorithm 1 (TPA1)** - This approach comes from employing the token population algorithm described in Fig. 3. The primary idea behind this algorithm is to simplify tokens by substituting lengthy permissions lists with overlapping roles. In order to accomplish this, the role with the maximum overlap must be identified. TPA1 achieves this by iterating through all the roles for each iteration in the main loop.
3) **Token Population Algorithm 2 (TPA2)** - This approach is similar to TPA1, but with the main difference being how the maximum overlap is calculated. TPA2 employs a priority queue to calculate the overlap, where the roles and their corresponding overlap counts are stored. Furthermore, the priority queue is sorted in descending order based on the overlap count, ensuring that the role with the maximum overlap remains positioned in the front of the queue.

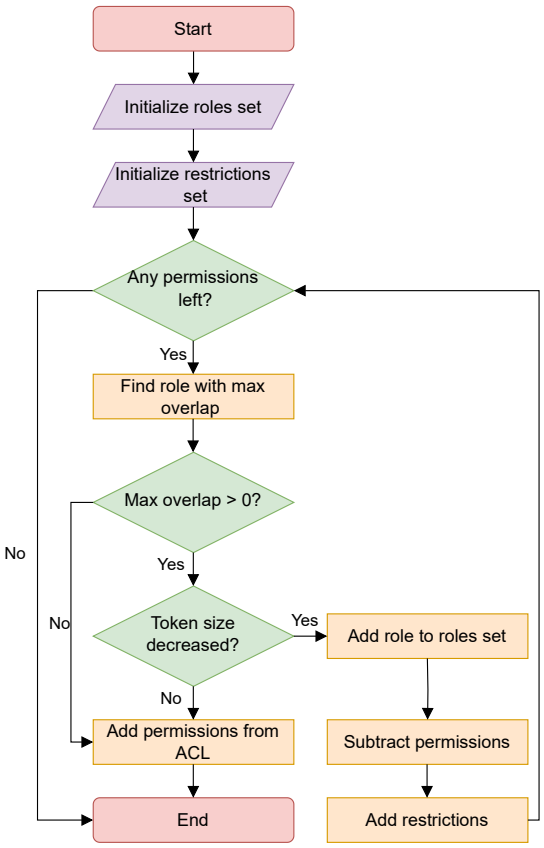

Fig. 3: Token population algorithm diagram.

IV. Demonstration

In this section, the implemented AS is demonstrated and experimentally evaluated. The main objective of the experiment is to measure the execution time, memory consumption,

and the size of the token for the three approaches of the token population.

A. Modular automation use case scenario

The implementation is integrated into a modular automation [18] system represented using a simulation of a modular ice cream factory. The system is illustrated in Fig. 4 and further described in [19], [20]. The system consists of six module controllers, which serve as resource servers in our scenario, and one orchestration unit, which issues high-level commands to the module controllers according to an SFC-formulated recipe. The orchestrator serves as a client in our scenario.

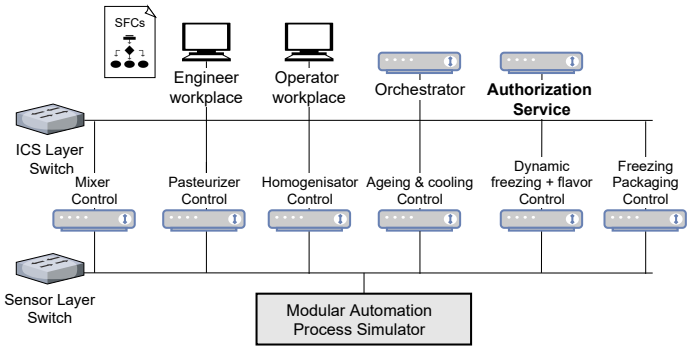

Fig. 4: Overview of the modular ice-cream factory use case.

B. Experimental setup

The ice-cream factory scenario is used as a proof of concept of the implementation, however, the recipes and roles used in the system are quite simple, and therefore response-time and memory consumption related to the AS is very small.

In order to perform meaningful scaling experiments, variable lists of synthetically generated servers along with their corresponding permissions are added to the ACL and RBAC lists. Execution time and memory consumption experiments involve the continuous addition of servers in increments of 50, ranging from 50 to 300. In the token size experiment, permissions are continuously added in increments of 50, ranging from 50 to 300. These permissions are manually generated and added to both ACL and RBAC lists. It should be noted that certain permissions overlap between the ACL and RBAC lists, while others differ. The selection and assignment of these permissions, whether identical or different, are determined randomly. Furthermore, 10 roles were incorporated into the RBAC lists for all experiments.

To ensure a reliable statistical average, each experiment was conducted 5 times for each different server/permissions load.

Experiments were conducted on Windows 10 operating system and implemented using the C# programming language. The System.Diagnostics.Stopwatch class from the .NET Framework is used in order to measure the execution time and memory consumption of the token generation process.

The experimental evaluation takes place on a computer system that features an Intel Core i5-10310U CPU, 16 GB RAM, and a 256 GB SSD storage device.

V. Results

A. Execution time

The purpose of the measurements is to determine the time it takes for the AS to generate the access token, i.e., to measure the time taken from when the AS first receives a request for a token, until it returns a fully populated token to the client.

The results are presented in Fig. 5, where the x-axis represents the number of servers, and the y-axis represents the average execution time in milliseconds (ms).

The baseline execution times are notably small, as expected due to the low complexity resulting from the absence of the token population algorithm.

The average execution time for TPA1 is higher than the one for the baseline. A detailed inspection of the results indicates a notable incremental pattern in the execution time as the number of servers increases. Compared to TPA1, TPA2 demonstrates a better performance on server loads of 250 and 300. However, considering smaller server loads it demonstrates slightly increased execution time.

Table I represents the standard deviation (σ) for execution time results in ms. The baseline configuration exhibits consistent standard deviation values, while the TPA1 and TPA2 configurations show varying standard deviation values for different server quantities.

TABLE I: Standard deviation (σ) for execution time, all numbers in ms.

Number of servers	6	50	100	150	200	250	300
Baseline	2.22	0.80	0.77	2.38	2.23	0.80	2.75
TPA1	2.07	4.49	3.81	2.44	6.35	43.78	27.68
TPA2	0.80	5.83	6.12	12.62	23.22	8.40	45.99

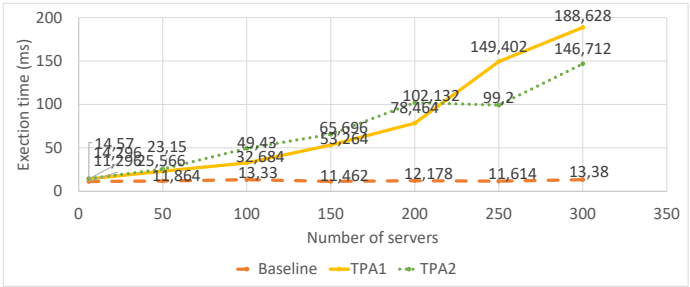

Fig. 5: Results on execution time experiment.

B. Memory consumption

This experiment is conducted in order to determine the amount of memory the AS consumes when handling the access token. Therefore, the memory usage is measured from the moment the AS initially receives a token request, up until the point when it completely populates that token. The results of memory consumption are represented in Fig. 6, where

the x-axis represents a load of added servers and the y-axis represents the memory consumption in Megabytes (MB). The results for the baseline show that it has low memory requirements and can operate efficiently, even when a load of 300 servers is considered.

The average memory consumption for TPA1 is higher than the one from the baseline. Results show that memory consumption starts to raise noticeably when the number of servers is greater than 150. Moreover, when considering the load of 150 servers or less, the memory consumption is similar to the baseline.

Memory consumption for TPA2 starts to drastically increase when considering more than 150 servers. On the other hand, when considering less than 150 servers, the memory consumption is similar to the TPA1 and the baseline approach. Additionally, for the load of 250 and 300 servers, TPA2 demonstrated slightly decreased memory consumption than TPA1.

Table II displays the standard deviation (σ) values for memory consumption in megabytes (MB). The standard deviation in the baseline scenario remains relatively low and stable across the range of server numbers. As the number of servers increases, the standard deviation for memory consumption in the TPA1 scenario also generally increases. The TPA2 scenario exhibits a more varied trend in standard deviation values. For smaller server numbers (150 and less), the standard deviation remains relatively low. However, as the number of servers goes over 150, the standard deviation increases.

TABLE II: Standard deviation (σ) for memory consumption, all numbers in MB.

No. of servers	6	50	100	150	200	250	300
Baseline	0.011	0.019	0.034	0.012	0.054	0.015	0.025
TPA1	0.034	0.091	0.101	0.401	7.23	4.12	4.89
TPA2	0.074	0.203	0.381	0.208	9.665	13.204	10.775

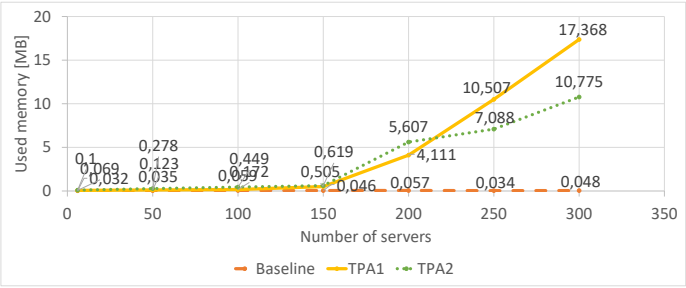

Fig. 6: Results on memory consumption experiment.

C. Token size

This experiment focuses on the size of the token provided by the AS. The token's size is measured as the sum of the number of permissions, restrictions, and roles that are encoded into it. In simpler terms, it represents the number of items in access token claims. The objective of the token population algorithm is to create the smallest possible token. This is crucial in the overall performance of the system since the tokens with more content require more memory and more execution time. Additionally, the obtained results are theoretical, and encoding this many permissions into a token would exceed the actual maximum size of the token.

The average sizes of ten tokens were observed, and their average sizes for each permissions load are represented in Fig. 7. The x-axis represents the number of permissions in the initial ACL, while the y-axis represents the size of the token (number of items in access token claims).

It is evident that for the load of 50 permissions, the token size remains the same for all of the variations. The results indicate a similarity between TPA1 and TPA2, without any evident differences. The data suggests that the noticeable difference between the baseline and TPA1/TPA2 can be seen when the load of 150 permissions is observed. Moreover, as the permissions load increases, the effectiveness of TPA1/TPA2 is becoming more noticeable.

The incremental pattern in efficiency that both TPA1 and TPA2 demonstrated is to be expected since the larger the number of permissions is, the larger the chance for the overlapping permissions that will be changed for the roles.

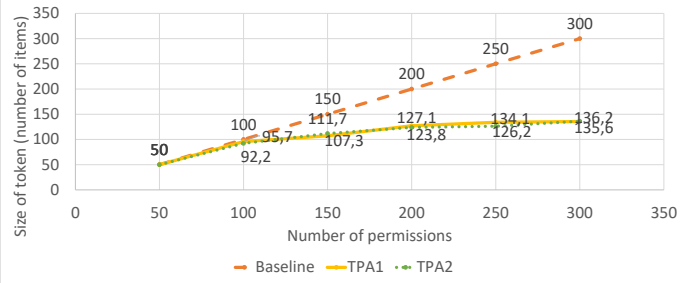

Fig. 7: Result on token size experiment.

VI. Discussion

The results of the experimental evaluation indicate that the utilization of token population algorithms directly impacts execution time, memory consumption, and token size. The baseline approach, which directly encodes ACL permissions into the token, provides the shortest execution time and lowest memory consumption, but results in larger token sizes, at least for complex scenarios. Contrary, the token population algorithms reduce the token size, at the cost of increased execution time and memory consumption. Moreover, the evaluation highlighted the trade-offs involved in choosing between token population algorithms and the baseline approach. When deciding whether to utilize the algorithm, it is important to take into account the trade-off between smaller token sizes and acceptable levels of execution time and memory consumption.

It is crucial to weigh the trade-offs when considering the impact of large tokens for a memory/CPU-constrained resource server. Issuing larger tokens could simplify the process of populating the token by encoding more information directly into the token. This could potentially lead to reduced CPU usage

and faster policy inference times, as complex algorithms would not be needed to reduce the token size. Conversely, in contrast, a smaller token reduces the memory and could be more suitable for memory-constrained environments. However, it might require more complex and CPU-intensive algorithms to populate and interpret the token, which could increase the policy inference time. For industrial applications, a reasonable policy inference time would need to be within a few 10s of milliseconds.

An approach for selecting the right encoding algorithm could be to use the size of the initial ACL to decide if attempts for optimizations are needed. For a small ACL, say below 20 permissions, there would be no gain in optimizing token size.

The evaluation performed within this work is limited to focuses solely on the AS, i.e., it does not cover the entire architecture. Furthermore, the data used to scale up the evaluation is synthetic, and may not faithfully represent realistic scenarios. The algorithm's efficiency in minimizing total token size is directly linked to the degree of similarity between permissions of the ACL and the role-permissions for the resource server. This similarity is randomly assigned within the synthetic data employed in the experiments conducted within this study.

VII. Conclusions

This work presents the implementation and evaluation of an AS that uses knowledge of executing workflows in order to make policy decisions. We have shown that the AS performs its assigned task in the context of a simulated ice cream factory, and we have experimentally evaluated a set of quality metrics related to three token encoding approaches, indicating the scalability properties of the implementation.

In conclusion, populating the token without the usage of a token population algorithm reduces the memory consumption and execution time, but it results in a larger token, which could lead to increased memory consumption and execution time once the token is issued.

The selection of approach should be based on a trade-off between the need for a smaller token size and the acceptable levels of execution time and memory consumption in the AS, possibly using an adaptable algorithm that can select an algorithm based on the initial ACL size or content.

Multiple directions for further research and development can be envisioned, e.g., validation of the access control enforcement architecture in real-world ICS environments or the exploration of different token population algorithms. In this work, we use knowledge of running workflows for the policy decisions, but other approaches are also possible, such as using logical conditions on attributes for creating the initial ACL. Formally investigating threats, vulnerabilities and possible attacks on the suggested approach is another important future direction of the work.

Acknowledgements

This work is supported by ABB AB; the industrial postgraduate school Automation Region Research Academy (ARRAY), funded by The Knowledge Foundation; and the Horizon 2020 project InSecTT. InSecTT (www.insectt.eu) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876038. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Sweden, Spain, Italy, France, Portugal, Ireland, Finland, Slovenia, Poland, Netherlands, Turkey.[3]

References

[1] J. Weiss, *Protecting industrial control systems from electronic threats*. Momentum Press, 2010.

[2] A. Sigov, L. Ratkin, L. A. Ivanov, and L. D. Xu, "Emerging enabling technologies for industry 4.0 and beyond," *Information Systems Frontiers*, pp. 1–11, 2022.

[3] C. Zanasi, F. Magnanini, S. Russo, and M. Colajanni, "A zero trust approach for the cybersecurity of industrial control systems," in *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, vol. 21, pp. 1–7, 2022.

[4] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, aug 2020.

[5] R. Lepro, "Cardea: Dynamic access control in distributed systems," technical report, NASA, 2004.

[6] T. W. Shinder, Y. Diogenes, and D. Littlejohn Shinder, "Chapter 7 - controlling access to your environment with authentication and authorization," in *Windows Server 2012 Security from End to Edge and Beyond*, pp. 187–210, Boston: Syngress, 2013.

[7] K. Knorr, "Dynamic access control through petri net workflows," in *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*, pp. 159–167, 2000.

[8] M. Uddin, S. Islam, and A. Al-Nemrat, "A dynamic access control model using authorising workflow and task-role-based access control," *IEEE Access*, vol. 7, pp. 166676–166689, 2019.

[9] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," in *IEEE Intl. Conference on Web Services*, vol. 2005, pp. 561–569, 2005.

[10] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," tech. rep., NIST, 2014.

[11] B. Leander, A. Čaušević, T. Lindström, and H. Hansson, "Access control enforcement architectures for dynamic manufacturing systems," in *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, pp. 82–92, 2023.

[12] B. Leander, A. Čaušević, H. Hansson, and T. Lindström, "Toward an ideal access control strategy for industry 4.0 manufacturing systems," *IEEE Access*, vol. 9, pp. 114037–114050, 2021.

[13] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)." RFC 7519, May 2015.

[14] F. Martinelli, O. Osliak, P. Mori, and A. Saracino, "Improving security in industry 4.0 by extending OPC-UA with usage control," in *15th Intl. Conference on Availability, Reliability and Security*, ACM, 2020.

[15] V. Watson, J. Sassmannshausen, and K. Waedt, "Secure granular interoperability with opc ua," in *INFORMATIK 2019: 50 Jahre Gesellschaft für Informatik – Informatik für Gesellschaft (Workshop-Beiträge)*, (Bonn), pp. 309–320, Gesellschaft für Informatik e.V., 2019.

[16] "IEC 62541 OPC unified architecture, rev 1.05," standard, International Electrotechnical Commission, Geneva, CH.

[17] "IEC 61131-3:2013 Programmable Controllers - Part 3: Programming Languages," standard, IEC, 2013.

[18] J. Ladiges *et al.*, "Integration of modular process units into process control systems," *IEEE Transactions on Industry Applications*, vol. 54, pp. 1870–1880, March 2018.

[19] B. Leander, T. Marković, A. Čaušević, T. Lindström, H. Hansson, and S. Punnekkat, "Simulation environment for modular automation systems," in *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, October 2022.

[20] B. Leander, T. Marković, and M. L. Ortiz, "Enhanced simulation environment to support research in modular manufacturing systems," in *IECON 2023–49th Conference of the IEEE Industrial Electronics Society*, IEEE, October 2023.

[3]The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.