Verification of Policies in Human Cyber-Physical Systems: the Role and Importance of Resilience

Antonios Gouglidis
School of Computing and Communications
Lancaster University
Lancaster, United Kingdom
a.gouglidis@lancaster.ac.uk

David Hutchison
School of Computing and Communications
Lancaster University
Lancaster, United Kingdom
d.hutchison@lancaster.ac.uk

***Abstract*—Cyber-physical systems (CPS) are characterised by interactions of physical and computational components. A CPS also interacts with its operational environment, and thus with other entities including humans. Humans are an important aspect of human CPS (HCPS) since they are responsible for using (e.g., administering) these types of system. Such interactions are usually expressed though access control policies, which in many cases (e.g., when performing critical operations) are required to support the property of resilience to cope with challenges to the normal operation of the HCPS. In this paper, we pinpoint the importance of resilience as a property in access control policies and we describe a mechanism to conduct its formal verification. Finally, we identify potential future directions in the verification of access control properties, complementary to resilience.**

***Index Terms*—access control, autonomy, model checking, policy, resilience, security, verification**

I. Introduction

Cyber-physical systems (CPS) are gaining considerable attention, now more than ever before, as the result of several advancements in engineering and sciences. Although there is an abundance of definitions for CPS, they all appear to converge to a single definition which states that a CPS is a network of both physical and computational components, co-engineered to interact together [1]. Examples of new technologies often classified as CPS are the Internet of Things (IoT), Industrial Internet, Smart Cities, etc. A CPS also interacts with its operational environment, and thus with other entities including, crucially, humans. Humans are usually part of such an environment since they are required to control a CPS, consume its output, and so on. We call these extended interactions between a human CPS (HCPS). Although there are several aspects that can be investigated, in this paper we are mainly interested in the concepts of access control policies and resilience since they are both considered to be of vital importance in the design of a HCPS [1].

The importance of access control in HCPS led to research in several directions, one of them being the investigation of properties related to the security offered by policies, e.g., secure inter-operation [2]. However, little attention has been paid to resilience as a property in access control policies. Resilience in access control is conceived as the ability of a system not to restrict, but to enable access to resources [3]. Most of the research work in this context is initiated around the *'resiliency checking problem'*, which examines whether a given resilience policy is satisfied by an access control state. This problem has been investigated from a generic point of view [3], and thus the proposed approaches are agnostic to the actual types of policy implemented by an underlying model. Additional research on the resiliency checking problem was performed to investigate the time complexity introduced by the various parameters used in it [4]. Moreover, the *'resiliency checking problem'* is shown to have a connection with the *'work-flow satisfiability problem'* in [5], with the latter being investigated extensively in the literature, e.g., in [6]–[8] among others. Further information on work-flow management systems and on how to model and enforce resilience policies is available in [9], [10].

The aim of this paper is to present a technique for the verification of resilience policies in a HCPS. To achieve this, we describe an automated technique for conducting formal verification (i.e., model checking) and elaborate on the verification of resilience policies, which we have previously presented in [11]. Furthermore, we provide a list of potential future directions, which indicate the need for investigating multiple properties, the definition of strategies to resolve conflicts among properties, and finally, the need to conduct additional checks to ensure completeness with regards to property verification.

II. Formal Verification

In this section, we provide brief information on model checking, which is a formal verification technique. Formal verification considers the use of applied mathematics for modelling and analysing systems, and its aim is to ensure the correctness of a system with mathematical rigour. Specifically, in model-based verification techniques, such as model checking, the possible states of a system are described through a transition system (TS). A set of specifications is also defined on the basis of linear-time properties. Model checking conducts an exhaustive exploration of all the possible states of a system, and checks whether the defined properties hold for a given state in the model [12].

The research presented in this paper is sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC) via grant agreement no. EP/L020009/1: Towards Ultimate Convergence of All Networks (TOUCAN).

Linear-time properties are usually classified into *'safety'* and *'liveness'* properties. Safety properties can be characterised as *'nothing bad should happen'* and liveness as *'something bad never happens'*. A *'bad'* situation is an undesirable state of the system – assuming a Smart City scenario with *'smart'* traffic lights, a *'bad'* situation is when the red, amber, and green lights are all on or all off. On the contrary, liveness properties require some progress – they are interpreted as *'something good will happen'* in the future [12].

The use of temporal logic, apart from providing a language for the property specification of policies, will eventually underpin the mathematical foundation used to formally verify access control policies. This requires the definition of a language for expressing polices and a TS able to describe the behaviour of the access control model, and thus for properties to be verifiable for the model.

We consider AP to be a set of atomic propositions (e.g., with α, β, $\ldots$ elements of AP). The set of propositional logic formulae over AP is inductively defined as:

- $true$ is a formula;
- Any atomic proposition, which is element of AP is a formula;
- If Φ, Φ_1 and Φ_2 are formulae, then are $(\neg\Phi)$ and $(\Phi_1 \wedge \Phi_2)$;
- Nothing else is a formula.

We say that the conjunction operator $\wedge$ binds stronger than the derived binary operators, such as that of disjunction, implication, etc. Specifically, we define the former two as in the following: $\Phi_1 \vee \Phi_2 := \neg(\neg\Phi_1 \wedge \neg\Phi_2)$ and $\Phi_1 \rightarrow \Phi_2 := \neg\Phi_1 \vee \Phi_2$, respectively. The $\rightarrow$ means *'imply'*.

We also assume the following notation regarding the associativity and commutativity law for disjunction and conjunction: $\bigwedge_{1\leq i\leq n} \Phi_i$ for $\Phi_1 \wedge \ldots \wedge \Phi_n$ and $\bigvee_{1\leq i\leq n} \Phi_i$ for $\Phi_1 \vee \ldots \vee \Phi_n$. If $I = \emptyset$, then $\bigwedge_{i\in\emptyset} \Phi_i := true$ and $\bigvee_{i\in\emptyset} \Phi_i := false$.

Then, we consider the $evaluation$ of atomic propositions. This is done by assigning a truth value to each of them, i.e., a function $\mu : AP \rightarrow \{0,1\}$, where 0 is $false$ and 1 is $true$. The $\rightarrow$ means *'maps to'*. Therefore, a *s*atisfaction relation $\models$ indicates the evaluations μ for which a formula Φ is $true$. Formally, it is written as:

- $\mu \models true$
- $\mu \models \alpha \iff \mu(\alpha) = 1$
- $\mu \models \neg\Phi \iff \mu \not\models \Phi$
- $\mu \models \Phi \wedge \Psi \iff \mu \models \Phi$ and $\mu \models \Psi$

Further on, we define the access control rule, the access control property, and the transition system of an access control model. Here we use the Computation Tree Logic (CTL) in order to specify policy properties. Linear-time Temporal Logic (LTL) could alternatively be used since we do not take advantage of the different expression level of neither CTL or LTL in our defined properties [13].

With regard to CTL, the prefixed path quantifiers assert arbitrary combinations of linear-time operators. Hence, we use the universal path quantifier $\forall$ that means *'for all paths'*, and the linear temporal operators $\Box$ and $\Diamond$ that mean *'always'* and *'eventually'*, respectively. Furthermore, we use the temporal modalities $\forall\Box\Phi$ representing *invariantly* Φ, and $\forall\Diamond\Phi$ representing inevitably Φ, where Φ is a state formula.

Definition 1: An access control rule is an implication of type '$c \rightarrow d$', where constraint c is a predicate expression, which when $true$ implies the permission decision d. The $\rightarrow$ means *'imply'*.

Definition 2: An access control property p is an implication formula of type '$b \rightarrow d$', where the result of the access permission d depends on $quantified$ predicate b.

Definition 3: A transition system TS is a tuple (S, Act, δ, i_0) where

- S is a set of states, e.g., $S = \{Permit, Deny\}$;
- Act is a set of actions;
- δ is a transition relation where $\delta : S \times Act \rightarrow S$;
- $i_0 \in S$ is the initial state.

The p in Definition 2 is expressed by the proposition $p : S \times Act^2 \rightarrow S$ of TS, which can be collectively translated in terms of logical formula.

The behaviour of the system is defined by the access control rules, and they function as the transition relation δ in TS. Thus, by representing an access control property using the temporal logic formula p, we can assert that model TS satisfies p by $TS \models \forall\Box(b \rightarrow \forall\Diamond d)$. Property $\forall\Box(b \rightarrow \forall\Diamond d)$ is a response pattern such that d responds to b globally (b is the cause and d is the effect) [14].

III. Verification of Resilience

Based on the theory provided in the previous section, resilience can be characterised as a *'safety'* property of a system. In this section, we elaborate on the notion of resilience policies, and discuss how this could be interpreted in the context of an access control model. Resilience policies are defined in [3]. Specifically, a resilience policy is defined as the tuple of $ResiliencePolicy\langle P, s, d, t\rangle$, where P is the set of permissions, $s \geq 0$, $d \geq 1$ and $t \in N^+$ or $t = \infty$. Thus, a resilience policy is satisfied in an access control state *'if and only if upon removal of any set of s users, there still exist d mutually disjoint sets of users such that each set contains no more than t users and the users in each set together are authorised for all permissions in P'* [3]. The construction of a resilience policy is also known in the literature as the *'resiliency checking problem'* [4], [3]. Specifically, given a resilience policy tuple $ResiliencePolicy\langle P, s, d, t\rangle$ the solution provides an answer to the existence of binary relation between users U and permissions P, i.e., $UP \subseteq U \times P$ [3], or between users U and their authorised resources R, i.e., $UR \subseteq U \times R$ [4]. In general, permissions are considered to be operations on objects. Assuming a system that operates in a critical infrastructure, we may have the following operations of a CPS device: *'monitor screen'*, *'start system'*, *'stop system'*, *'disable alarm'*, and *'change set points'*, and thus we set P = $\{$*Supervisor, Manager*$\}$. Given P, we may have the following values for the rest of the resilience policy parameters: $s = 1$, $d = 1$, and $t = 1$. Specifically, $s = 1$ indicates that we want

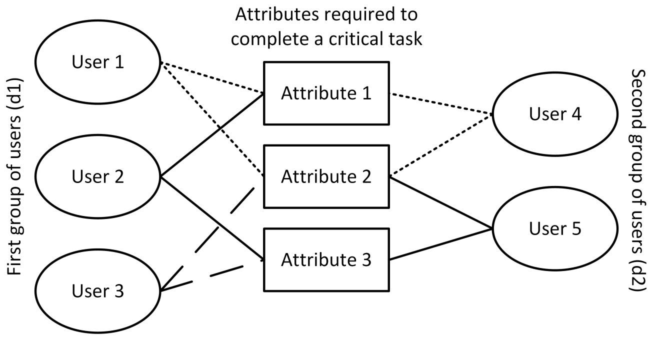

Fig. 1. Example of a resilience policy

the policy to be resilient to the absence of any (one) user, $d = 1$ indicates that we require one set of users such that users in that set together possess all permissions; and, $t = 1$ since there is a single user that has all the permissions [3].

The definition of a resilience policy requires initially a careful definition of the different critical tasks in a HCPS and subsequently identification of the main users and assigned permissions required to successfully complete these tasks. As mentioned already, this process can be performed during the early stages of the design of a system. Nevertheless, users and policies may change in a system, i.e., certain policies may be altered, deleted or new policies may be introduced. Therefore, these operations may introduce disruptions in an already existing resilience policy. Designing these policies from scratch may not be a viable solution, especially in the context of a HCPS, where systems must operate in an uninterrupted manner. Hence, administrators or operators in such environments may require to verify at any time the resilience offered by the active set of policies in their operational environment. Such an approach may also lead to reducing the overall complexity imposed by solving the resiliency checking problem from scratch.

In Fig. 1, we consider an attribute-based access control model (ABAC) as defined in [11] to provide an example of a resilience policy. We also consider a critical task T in a HCPS. In order to successfully accomplish the critical task, the users have to be collaboratively authorised for all three attributes. In this example, we consider two groups of users, where the first group includes the following users and assigned attributes: $User1 \times \{Attribute1, Attribute2\}$, $User2 \times \{Attribute1, Attribute3\}$, $User3 \times \{Attribute2, Attribute3\}$; and the second group includes the following users and attributes: $User4 \times \{Attribute1,$ $Attribute2\}$, $User5 \times \{Attribute2,$ $Attribute3\}$. In the context of an industrial control system, the above attributes could be equivalent with: $Attribute1 \equiv$ (monitor a device), $Attribute2 \equiv$ (start or stop a device), and $Attribute3 \equiv$ (maintain a device). Thus, in case of a device malfunction, operators (i.e., users of the HCPS) shall be in position to monitor and acknowledge the problem, stop the faulty device, maintain the device, and finally, start the device.

Therefore, assuming the response property pattern defined in Section II, we can define resilience specifications and check their satisfiability using formula 1.

$$TS \vDash \forall\Box\Big(\bigwedge_{1\leq i\leq n} !sub_n \bigwedge_{0\leq i\leq m} attr_m \bigwedge_{1\leq i\leq k} !sub_k \rightarrow \forall\Diamond Deny\Big) \tag{1}$$

where TS is the resilience ABAC policy transition system, sub_n, sub_k are subjects (e.g., users) of a system, $sub_n \neq sub_k$, $\{sub_n\}\times\{attr_m\} \in Act$, and $Deny \in S$ is the permission decision. In relation to the resilience policy tuple, i.e., $\langle P, s, d, t\rangle$, sub_n is mapped onto the set of users s that are considered to be absent; $attr_m$ refers to the attributes assigned with a user s and represent permissions required to perform a task; and, sub_k refer to the mutual disjoint set of users expressed by d. The t parameter can be introduced implicitly by adding specifications following formula 1.

IV. Future Directions

Although resilience is undoubtedly an important property of access control policies, it is one of several that have to be ensured in a HCPS access control policy. In case the HCPS is a collaborative system, it may also be required to verify properties as secure inter-operation [2]. Secure inter-operation in collaborative systems is required for secure collaboration among participating parties such that the principles of autonomy and security can be guaranteed. The principle of autonomy states that if an access is permitted by an individual system, it must also be permitted under secure inter-operation. The principle of security states that if an access is denied by an individual system, it must also be denied under secure inter-operation. From the above definitions it is obvious that conflicts may occur when trying to ensure both security and autonomy at the same time. Furthermore, when considering additional properties such as resilience, further conflicts may arise. Therefore, it is evident that although the verification of the individual properties may not raise any concerns, ensuring multiple properties may be problematic. Thus, conflict resolution strategies should be investigated to resolve potential conflicts. Nevertheless, these strategies may vary depending on the type of process or service offered by the HCPS.

All properties mentioned above are characterised as *'safety'* ones. This implies that verifying their absence will ensure that nothing bad should happen. However, it is also important to investigate *'liveness'* properties in HCPS. These are properties that complement *'safety'* ones and require some progress, i.e., properties that are violated in infinite time [12]. To the best of our knowledge, liveness properties are not investigated adequately in HCPS access control policies.

Finally, coverage and confinement are considered to be important aspects of verification. The formal verification of *'safety'* or *'liveness'* results in ensuring the logic integrity of the access control policies against them. Nevertheless, for reasons of completeness, the policies should be verified against coverage and confinement faults as well [15]. With regard to coverage, mutated models of the original model should be checked. The mutated models will include changes in the logic of policies, i.e., a rule r will change from '$c \rightarrow d$' to '$c \rightarrow \neg d$'. If the *'safety'* or *'liveness'* properties are satisfied against both the original and the mutated models, then this is an indication that the verified properties do not cover all the policies in the defined model. With regard to confinement, the model should be verified against an extended set of *'safety'* or *'liveness'* that may include negative expressions of existing specifications, i.e., specifications of the form $\forall\Box(b \rightarrow \forall\Diamond d)$ will have to change in $\neg\forall\Box(b \rightarrow \forall\Diamond\neg d)$. Confinement checking should discover the discrepancy of the specified *'safety'* or *'liveness'* properties and the *'safety'* or *'liveness'* properties the access control policy author intend. The rationale is that if the model does not satisfy the modified specifications, then there are access permissions that may leak through the *'safety'* or *'liveness'* properties [15].

V. Conclusion

Human CPS may be considered to be an extension of CPS. These systems are known to underpin technologies such as the Internet of Things which are becoming increasingly important in critical infrastructures, on which our society depends. In that context, resilience is a concept of vital importance. To introduce resilience by design in access control policies, we described a formal verification technique that may facilitate its adoption. Existing toolchains described in [2], [11], [15] may be used to implement the approach described in this paper. Finally, we provided three main future directions when considering the verification of properties in HCPS, and we anticipate these directions to provide interesting multi-disciplinary insights in both industry and academia, and to stimulate further research in this important field of study.

References

[1] NIST, "Framework for cyber-physical systems," Cyber Physical Systems Public Working Group, 2016. [Online]. Available: https://goo.gl/FcL1rZ

[2] A. Gouglidis, I. Mavridis, and V. C. Hu, "Security policy verification for multi-domains in cloud systems," *International Journal of Information Security*, vol. 13, no. 2, pp. 97–111, 2014.

[3] N. Li, Q. Wang, and M. Tripunitara, "Resiliency policies in access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 4, p. 20, 2009.

[4] J. Crampton, G. Gutin, S. Pérennes, and R. Watrigant, "A multivariate approach for checking resiliency in access control," *arXiv preprint arXiv:1604.01550*, 2016.

[5] J. Crampton, G. Gutin, and R. Watrigant, "Resiliency policies in access control revisited," in *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*. ACM, 2016, pp. 101–111.

[6] D. Cohen, J. Crampton, A. Gagarin, G. Gutin, and M. Jones, "Iterative plan construction for the workflow satisfiability problem," *Journal of Artificial Intelligence Research*, vol. 51, pp. 555–577, 2014.

[7] J. Crampton, G. Gutin, and D. Karapetyan, "Valued workflow satisfiability problem," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*. ACM, 2015, pp. 3–13.

[8] Q. Wang and N. Li, "Satisfiability and resiliency in workflow authorization systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 4, p. 40, 2010.

[9] V. Atluri and J. Warner, "Supporting conditional delegation in secure workflow management systems," in *Proceedings of the tenth ACM symposium on Access control models and technologies*. ACM, 2005, pp. 49–58.

[10] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 1, pp. 65–104, 1999.

[11] A. Gouglidis, V. C. Hu, J. S. Busby, and D. Hutchison, "Verification of resilience policies that assist attribute based access control," in *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*. ACM, 2017, pp. 43–52.

[12] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of model checking*. MIT press, 2008.

[13] R. B. Krug, "CTL vs. LTL," Presentation, May 2010. [Online]. Available: http://www.cs.utexas.edu/users/moore/acl2/seminar/2010.05-19-krug/slides.pdf

[14] SAnToS Laboraroty, "Specification patterns, Responce property pattern," 2012. [Online]. Available: http://patterns.projects.cis.ksu.edu/documentation/patterns/response.shtml

[15] V. C. Hu, R. Kuhn, and D. Yaga, "Verification and test methods for access control policies/models," *NIST Special Publication*, vol. 800, p. 192, 2017.