Addressing Security Properties in Systems of Systems: Challenges and Ideas

Miguel Angel Olivero[1,2] [0000-0002-6627-3699], Antonia Bertolino[1] [0000-0001-8749-1356], Francisco José Dominguez-Mayo[2,3] [0000-0003-3502-8858], María José Escalona[2,3] [0000-0002-6435-1497], and Ilaria Matteucci[4] [0000-0002-5936-8470]

[1] Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche.
[2] Web Engineering and Early Testing (IWT2) Research Group, Universidad de Sevilla.
[3] Computer Languages and System Department, Universidad de Sevilla.
[4] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche.
miguelangel.olivero@isti.cnr.it, antonia.bertolino@isti.cnr.it,
fjdominguez@us.es, mjescalona@us.es, ilaria.matteucci@iit.cnr.it,

Abstract. Within growing pervasive information systems, Systems of Systems (SoS) emerge as a new research frontier. A SoS is formed by a set of constituent systems that live on their own with well-established functionalities and requirements, and, in certain circumstances, they must collaborate to achieve a common mission. In this scenario, security is one crucial property that needs to be considered since the early stages of SoS lifecycle. Unfortunately, SoS security cannot be guaranteed by addressing the security of each constituent system separately. The aim of this paper is to discuss the challenges faced in addressing the security of SoS and to propose some research ideas centered around the notion of a mission to be carried out by the SoS.

Keywords: Mission-oriented modeling and testing, Security, System of Systems

1 Introduction

The challenge of governing the cooperation among a set of independent systems dynamically interconnected and working as a large complex system has been addressed by researchers since the early 90's [1] [2]. In recent years, this concept is referred to as a "System of Systems" (SoS) [3]. In their extensive review of SoS concepts and techniques, Nielsen and coauthors [3] provide several examples of domains where SoS becomes prevalent, including transportation networks, smart energy grids, and e-commerce applications, with emergency management remaining the most evident case in which SoS are extensively used [4].

A SoS aims at achieving global goals that would be infeasible for its constituent systems working in isolation. Such SoS goals have been named as *missions* [5]. The conceptual model of a mission drives the representation of the SoS emergent behavior, such as, among others, the involved tasks and constraints, the mission trigger, the executor systems, and so on [6]. Indeed, SoS missions are a key component when modeling or validating an SoS, as well as when defining its architecture.

Silva and coauthors have proposed mKAOS [5], which is both a mission-oriented language and an approach for modeling and designing SoS. The approach is based on the Goal Oriented Requirements Engineering, or GORE [7]. The mKAOS language extends KAOS/SysML (a language provided by the Object Management Group) and allows to assign a set of missions to each constituent system in the SoS. In this language, the sum of all joint works of the systems describes the functionality of the whole system.

However, the problem of modeling and addressing non-functional properties of SoS is not covered by mKAOS so far, and still remains largely unexplored [8]. In particular, concerning security, Ki-Aries and coauthors state that there exists "*no clear guidance or limited tool-support integrating different modelling elements to visualize and assess the SoS security consequences*" [9].

To address this need, we aim at modeling SoS security requirements in the context of SoS missions. Once the security requirements have been established, we also aim at validating the possible different SoS solutions, by means of an appropriate testing campaign. In fact, also concerning SoS security testing we identified a gap in current literature: to the best of our knowledge a specific approach addressing SoS security testing does not yet exist.

Summarizing, our research tackles security challenges on SoS and considers using a mission-oriented security modeling and testing approach that we refer to as Testing for Security in System of Systems (TeSSoS). We briefly introduced all steps that compose TeSSoS in [10] and stated the purposes of this method. In this paper, we lay the wider scene for such research, providing motivations, and discussing relevant challenges. In particular, we discuss the differences when addressing the security among different SoS architectures and how these challenges could be addressed. For completeness, we also include an outline of the on-going approach TeSSoS [10].

This paper is organized as follows: Section 2 revises existing SoS architectures by distinguishing how the different constituent systems organize themselves to achieve a common goal. It also discusses their particular security issues and describes the identified challenges for each SoS architecture. Section 3 discusses about the depicted scenarios and introduces the TeSSoS approach to assess the security in SoS that includes modelling and testing. Finally, the conclusions of our work are in Section 4.

2 Security Issues in Systems of Systems

Security is among the most relevant and critical features of SoS. It is a special concern for researchers in the military domain and in the Information Technology area as well [11]. The main challenge in analyzing and testing security properties in SoS derives from the non-compositional nature of security properties. In fact, guaranteeing the security of each constituent component of the SoS does not guarantee that the SoS is secure as a whole.

Indeed, one of the main aspects of SoS is that they are dynamically evolving. The constituents that compose the SoS may change at any time, or some new systems may be added to the environment or removed. Hence, when assessing the quality of a SoS and its evolution, it is important to consider any mechanism able to guarantee security

and avoid as many vulnerabilities and weaknesses as possible, for example when new systems join. the target SoS.
Every SoS has four general interdependency threats [12]:

(1) A constituent system failure;
(2) A constituent system impersonation;
(3) Communication channel failure;
(4) Communication channel infection.

Additionally, a fifth threat is introduced when constituents are sharing data. This vulnerability arises because by merging partial results coming from different constituent systems, more information becomes available and can be exploited for an SoS attack. Perhaps the constituent systems are sharing data, that, taken in isolation, are meaningless. However, when combining the set of data available from more systems, valuable information can be generated in a synergic way that can compromise the SoS.

In the literature four different architectures of SoS have been defined: Directed, Acknowledged, Collaborative, and Virtual [13] [14]. Figure 1 summarizes the process for categorizing the system according to its key features. By knowing about the existence of a central entity and the existence of guidelines, SoS can be organized in any of the four categories. The key factor that distinguishes one architecture from another is how they communicate and interact among them: the security issues that may affect an SoS vary depending on its architecture. In a similar way, testing each SoS is different depending on the architecture because of the nature, that require a different responsible for managing the security. In the remainder of this section, we discuss such security issues in the four different SoS architectures.

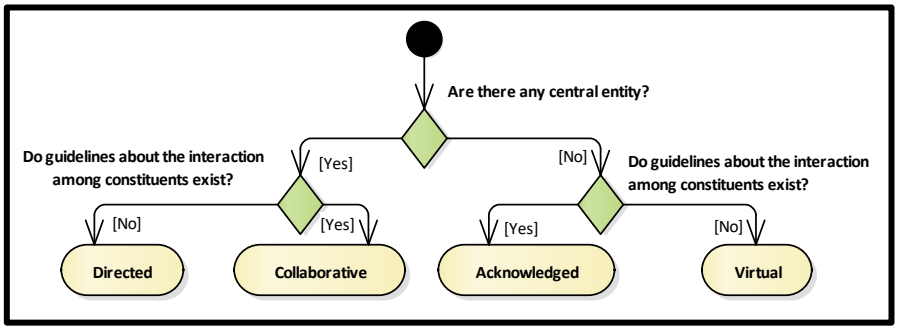

Fig. 1. SoS architecture decision tree

For the sake of understanding, using an airport as a fil rouge, for each architecture we supply different scenarios related to distinct subsystems of an airport as examples of SoS.

2.1 **Directed SoS**

Directed SoSs are managed by a central system that handles the success of the global purpose. Each constituent system is independent, but prioritizes the tasks commanded by the central. An example of a Directed SoS is the airport's system of surveillance, where each part, such as cameras infrastructures, or boarding pass scanners, are independent on their own when disconnected from the SoS, however these systems receive and execute commands according to precise guidelines when are integrated in the SoS.

Security testing in Directed SoS is the simplest case. It relies on the communication of each constituent system with the central one, which is responsible of security negotiation among constituent systems.

Indeed, by having a central entity that manages all the communications, security requirements and properties can be managed in a centralized way. The constituents can be centrally organized not only to achieve the final goal, but also to satisfy security requirements. The latter will require monitoring (in both, active and passive way) the communications among constituents, and the behavior of each constituent as well. Note that governing the set of security requirements of a Directed SoS may require a negotiation phase among the constituents.

The main challenge for the security when using this kind of architecture is to establish the common criteria that define the shared concept of security. In this sense, every system that would coordinate with the central entity must agree with the security requirements. Given the fact that every constituent system needs to communicate with the central entity, a strategy based on access control may be helpful to avoid unauthorized use of resources.

2.2 **Collaborative SoS**

An SoS is Collaborative when, even being coordinated by a central authority, the constituent systems retain self-control. The constituents are advised by the central system, but the final decision upon their actions is taken by the constituent systems themselves. Referring again to the Airport example, the landing track can be seen as an Collaborative SoS. The central system is the command tower, each system knows every other one; there are some passive systems, but there are others that maintain their independence, the planes.

As in the case of the Directed SoS, the central entity can select and coordinate the constituents in such a way that security requirements are satisfied by the emergent SoS. Conversely with respect to the Directed SoS case, the central entity is not able to monitor or correct potential insecure behaviors of the constituents.

Furthermore, the SoS lacks control on the behavior of each single constituent. The data shared with the constituent systems may be manipulated by functionalities out of the scope of the SoS, which introduces vulnerabilities on the privacy of the SoS data.

Challenges for this architecture include the ones defined for Directed SoS. However, since these systems retain independence, and are not strictly controlled by the central system, it would be possible for a constituent system to use information from other constituent systems for its own purposes, intentionally or unintentionally. This problem introduces the need of clearly stating which are the essential data each constituent

requires, to avoid providing data that are not strictly necessary. In security-related scientific literature the fact of not sharing more than strictly needed data, is known as non-disclosure or data sharing agreement. Nevertheless, in the event of an attack, the attacker could access data shared among the different constituents and reconstruct sensitive information about the SoS, that may be used to exploit its security in other attacks.

Testing the security in Collaborative SoS is like the Directed SoS security testing since there is a central system that helps in the negotiation of the security among the constituent systems. However, despite the architecture is similar, the number of vulnerabilities to test on the SoS increases. This is produced in part by the need of trusting other constituents as well as testing the final purpose of the SoS. It is necessary to define mechanisms for testing the communication that each constituent system performs with other constituents, as well as the functionality or behavior of the systems when processing data from the SoS. These tests should aim at discovering if there are any potential data leaks that combined with other data can reveal information that jeopardizes the security of SoS.

2.3 **Acknowledged SoS**

Acknowledged SoS are not controlled by a central system, but they may abide by an agreement on performing certain tasks. Acknowledged SoS in the airport context could be transport services such as the taxi company, or the autobuses. There is no common entity that manages all of them, notwithstanding they know each other and know they are somehow cooperating to allow people timely reach their destinations.

Managing the security in this kind of architecture requires a distributed and decentralized organization. Each constituent oversees the mutual agreements and should cooperate and coordinate with the others. In this architecture, a security requirements negotiation phase is essential. Additionally, each constituent must guarantee to behave correctly, i.e., in a compliant way with respect to the set of agreed requirements.

Challenges from Directed and Acknowledged architectures are also present in this one. However, given the fact that there is not a central entity to coordinate, but a common goal to achieve, individual interests of the systems may arise and create security vulnerabilities. Privacy could be also affected in this architecture. Constituent systems may change from a SoS to another according to their availability. When doing joint work, systems share data and functionalities, however, some systems may make an improper use of the collective data for their own or third parties' profit. Thus, lack of trust and/or lack of responsibilities among the constituent systems in an acknowledged architecture could become a considerable risk as for the shared data and functionalities.

Hence testing acknowledged SoS is more complex than in the previous cases. The lack of a central system helping in the negotiation of the security requirements makes an extensive coordination process necessary, in which each system shall conduct this negotiation on its own sake.

During a collaboration, the systems may generate a cascade problem [15], which may occur when a system with high security levels is sharing data with a system with lower security. In other words, a different level of security among constituent systems,

causes that the system with lower security level become the weakest link in the SoS. At the time of working in this architecture, an extensive analysis of the constituent systems must be executed to detect the weakest systems and determine if any of them could create the cascade problem. The chain of systems may be extended to deeper levels, analyzing also other SoS on which each constituent system is also working.

2.4 **Virtual SoS**

Virtual SoS emerge in unpredictable ways, as an outcome of the results coming from individual systems. They are not coordinated by a central system and the systems may not even know that they are working for a global purpose. An example of this architecture in the airport context is a set of shops in the duty-free section. These shops do not know about each other if it is not necessary, and they do not have a common purpose to achieve.

Managing the security in this architecture needs a distributed and decentralized organization. There are no formal agreements among the shops, but they are providing services for clients who may combine the items that these different shops offer. This is the architecture that may present more difficulties when analyzing its vulnerabilities with respect to security, because there is neither a central entity that may guarantee security nor an agreement that describes which should be the correct behavior. On the other hand, exactly because they collaborate loosely, the vulnerabilities might have minimal impact on each single system on average. The trust is not considered on this architecture, and the purpose of the global mission does not conditionate the functionalities of each single constituent.

Security requirements cannot be easily tested in virtual SoS because there is uncertainty about how the constituent systems would communicate in the future, however it could be analyzed considering previous SoS collaborations. In SoS architectures, pieces of data from different systems may be put together and produce information that exposes the security of another. Despite this, Virtual architecture might provide the same challenges as the previous architectures and include an additional one, the inability of knowing what are those systems that could provide such pieces of data. To the best of our knowledge, no defensive mechanisms can be clearly defined for addressing this problem, but contingency plans can be defined to mitigate an exploitation of the security.

3 **Addressing security**

Our work aims at providing a method to address the security issues arising in the SoS architecture. The method, named Testing for Security in System of Systems (TeSSoS) [10], focuses in modeling the security requirements of the SoS and generating the test cases to evaluate the security. TeSSoS is an ongoing work that has been designed as a set of five stages. At each stage, guidelines are supplied to assess the security challenges emerging in the SoS under exam.

3.1 **Modeling Security of System of Systems**

Models in SoS are dynamic because an SoS is constantly evolving. Every new incoming system arriving into the SoS needs to be analyzed to keep some standard security level among the constituent systems.

To model the security in SoS, as in isolated systems, it is necessary to consider threats, vulnerabilities, weaknesses, attackers, and attacks that affect the assets. In this view, we are modeling the security and the synergic features of the constituent systems in the SoS. To address the modeling of the security we consider an SoS already modelled with its functional features. On this basis, to model the security properties, it is mandatory to analyze the communication among each constituent system, channels, and their contents, and study the activities that each system performs over these communication channels.

The first three stages in the TeSSoS approach target the SoS modeling and the security analysis. The first one is SoS Discovery that focuses on the SoS modeling, eliciting the constituent systems agreements, and defining the assets. In the second stage, called Red Requirements, the SoS model and its vulnerabilities are analyzed. Red Requirements were designed to allow reusing the modeled vulnerabilities to be addressed so these can be used as test cases. In this way, Red Requirements are also used to evaluate if vulnerabilities have been solved. As a result, a catalog of potential attacks is produced and written in Gherkin[1] language, which is ready for testing stages. Third, Blue Requirements supply counter-measures to avoid earlier identified vulnerabilities to succeed. Human training is also considered as a countermeasure to avoid attacks since the human factor is the one that affects the most security properties.

The SoS models, the detected vulnerabilities and proposed improvements are defined in the three first phases of TeSSoS. The method continues with systems development and the humans training. The Blue Requirements provide the catalog of User Stories ready to be developed for the development team of each constituent system according to their responsibilities in an agile environment [16]. However, some difficulties may arise when developing or training, since systems in the SoS can be managed by third parties.

3.2 **Testing the Security of System of Systems**

In the SoS context it is common to find third-party black-box systems among the constituent systems.

Given that security is a non-compositional feature, and the security of the SoS does not depend only on the security of the constituent systems, testing the security in SoS is not just testing the security on each single constituent system, but testing the communication among the constituents.

The TeSSoS approach includes a testing phase after the development and training. The design of the test cases is not necessary since it is possible to reuse the Red Requirements definition as test cases. This fact allows reducing testing phase just to execution and evaluation.

[1] https://cucumber.io/docs/reference#gherkin

However, since some constituent systems may be black-box systems, we can only rely on the behavior of these systems. The tester could for instance perform the same actions an attacker or an accidental user would do in a Penetration testing [17]. Another common security testing approach that behaves in this way is Fuzz testing [18] [19]. Fuzzing works by analyzing the output and behavior of the system under test when it is stimulated with random input. This testing technique can be applied with different perspectives by considering not only to randomly modify the content messages, but also the sequence order of such messages, or apply some kind of knowledge instead of full random generation. Alternatively, we could derive a model-based approach that is based on the Red requirements model using penetration testing on which the attacker behavior is replicated [20]. This testing strategy is carried out by the so-called Red Team, which simulates an attacker.

4 Conclusions

In this work we have reviewed the security challenges over the four possible architectures of an SoSs. For each identified architecture we provide some examples of how the joint work is managed using the environment of an airport.

Given their natures, each architecture has different security challenges and, according to their architecture, different approaches are identified to be addressed. To analyze the security, and to detect the potential vulnerabilities on the SoS, we introduce an ongoing work named TeSSoS. This proposal organizes a set of ordered stages that guide the process of analyzing the security in the SoS context through modeling and testing.

Considering the challenges that face the security described in this work, future work will focus in addressing the problem of modeling and testing security requirements for the SoS. To this end, the phases in the TeSSoS approach will be defined in detail with the challenges of the different architectures in mind.

Acknowledgments

This work has been partially supported by the GAUSS national research project (MIUR, PRIN 2015, Contract 2015KWREMX) and by the Spanish Ministry of Economy and Competitiveness (POLOLAS, TIN 2016-76956-C3-2-R)

References

1. J. D. Richardson and T. J. Wheeler, “An object oriented methodology integrating design, analysis, modelling, and simulation of systems of systems.” 4th Annual Conference on AI, Simulation and Planning in High Autonomy Systems, 1993, pp. 238-244.
2. D. J. Bodeau, “System-of-systems security engineering”. In Proc. 10th Annual Computer Security Applications Conference, 1994. (pp. 228-235).
3. C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska. “Systems of systems engineering: basic concepts, model-based techniques, and research directions.” ACM Computing Surveys (CSUR), 48(2), 18, 2015.

4. S. Liu. “Employing system of systems engineering in China's emergency management.” IEEE Systems Journal, 2011, 5.2: 298-308.
5. E. Silva, T. Batista, and F. Oquendo, “A mission-oriented approach for designing system-of-systems,” in SoSE, 2015, pp. 346–351.
6. E. Silva, E. Cavalcante, T. Batista, F. Oquendo, F. C. Delicato, P.F. Pires, “On the characterization of missions of systems-of-systems”. In: European Conference on Software Architecture Workshops. ACM, 2014. p. 26.
7. A. van Lamsweerde, “Goal-oriented requirements engineering: a guided tour,” Proc. Fifth IEEE Int. Symp. Requir. Eng., 2001, pp. 249–262.
8. V. Chiprianov, K. Falkner, L. Gallon, and M. Munier. “Towards modelling and analysing non-functional properties of systems of systems”. On SOSE, 2014 (pp. 289-294).
9. D. Ki-Aries, S. Faily, H. Dogan, and C. Williams. “Assessing System of Systems Security Risk and Requirements with OASoSIS. In ESPRE (pp. 14-20). IEEE. 2018.
10. M. A. Olivero, A. Bertolino, F. J. Dominguez-Mayo, M. J. Escalona, I. Matteucci. “Security Assessment of Systems of Systems” in SESoS, 2019.
11. T. Bianchi, D. S. Santos, and K. R. Felizardo, “Quality Attributes of Systems-of-Systems: A Systematic Literature Review,”. SESoS 2015, pp. 23–30, 2015.
12. C. Guariniello and D. DeLaurentis, “Communications, information, and cyber security in systems-of-systems: Assessing the impact of attacks through interdependency analysis,” *Procedia Comput. Sci.*, vol. 28, no. Cser, pp. 720–727, 2014.
13. W. G. J. Halfond, S. R. Choudhary, and A. Orso, “Penetration testing with improved input vector identification,” Proc. - 2nd International Conference Software Testing, Verif. Validation, ICST 2009, pp. 346–355.
14. J. S. Dahmann and K. J. Baldwin, “Understanding the current state of US defense systems of systems and the implications for systems engineering,” SysCon 2008, pp. 99–105.
15. J. D. Horton et al., “The cascade vulnerability problem,” *J. Comput. Secur.*, vol. 2, no. 4, pp. 110–116, 1993.
16. M. Cohn, “User stories applied: For agile software development.” Addison-Wesley Profession, 2004.
17. B. Beizer, “Black-box testing: techniques for functional testing of software and systems.” 1995.
18. B., Shanmugam; N. B. Idris. “Improved intrusion detection system using fuzzy logic for detecting anamoly and misuse type of attacks”. In *2009 ICSCPR* (pp. 212-217).
19. G. Tian-yang, S. Yin-sheng, and F. You-yuan. “Research on software security testing.” World Academy of Science, Engineering and Technology Issue, 69:647– 651, 2010.
20. Bacudio, A. G., Yuan, X., Chu, B. T. B., & Jones, M. “An overview of penetration testing”. *International Journal of Network Security & Its Applications*, 2011.