Security Assessment of Systems of Systems

Miguel Angel Olivero
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale della Ricerche
Pisa, Italy
Web Engineering and Early Testing Research Group
Universidad de Sevilla
Seville, Spain
miguelangel.olivero@isti.cnr.it

Antonia Bertolino
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale della Ricerche
Pisa, Italy
antonia.bertolino@isti.cnr.it

Francisco José Dominguez-Mayo
Computer Languages and Systems Department
Universidad de Sevilla
Seville, Spain
fjdominguez@us.es

María José Escalona
Computer Languages and Systems Department
Universidad de Sevilla
Seville, Spain
mjescalona@us.es

Ilaria Matteucci
Istituto di Informatica e Telematica
Consiglio Nazionale della Ricerche
Pisa, Italy
ilaria.matteucci@iit.cnr.it

Abstract— Engineering Systems of Systems is one of the new challenges of the last few years. This depends on the increasing number of systems that must interact one with another to achieve a goal. One peculiarity of Systems of Systems is that they are made of systems able to live on their own with well-established functionalities and requirements, and that are not necessarily aware of the joint mission or prepared to collaborate. In this emergent scenario, security is one crucial aspect that must be considered from the very beginning. In fact, the security of a System of Systems is not automatically granted even if the security of each constituent system is guaranteed. The aim of this paper is to address the problem of assessing security properties in Systems of Systems. We discuss the specific security aspects of such emergent systems, and propose the TeSSoS approach, which includes modelling and testing security properties in Systems of Systems and introduces the Red and Blue Requirements Specification concepts.

Keywords—Blue Requirements, Red Requirements, Security Assessment, System of Systems

I. Problem Statement

Nowadays the term "System of Systems" (SoS) has become a catchword to characterize the notion of a large complex system made by the interconnection of a set of independent and typically pre-existing constituent systems [1]. SoSs are not a novelty, actually, as their associated challenges are acknowledged and studied at least since the early 90's, e.g., [2] [3]. However, with the growing pervasiveness of software in all aspects of contemporary life, the huge computational resources made available by the Cloud, and the ubiquitous connectivity among the devices around us, the need and/or the opportunity of using SoSs emerge in many domains, oftentimes even without an explicit recognition of the SoS as such. The attractiveness of SoS architectures descends from the fact that the SoS collective behavior can achieve goals that would be infeasible by having the constituent systems working in isolation. In the literature such collective goals are referred to as the *SoS missions*. Explicitly identifying and modeling a SoS mission may provide key guidance for SoS design and validation. In fact, a mission conceptual model can help in representing and relating the main elements of the SoS emergent behavior, such as, among others, the involved tasks and constraints, the mission trigger, who are the executor systems, and so on [4].

In this direction, after having identified a gap in the literature about how missions should be modeled, Silva et al. have recently proposed the mKAOS mission-oriented language and approach for modeling and designing SoS [5]. However, the problem of modeling and addressing non-functional properties (NFPs) of SoS [6] remains largely unexplored. While a mission success may be affected by poor resulting global performance, security, or other NFPs, such non-functional aspects are hardly measurable or predictable in a SoS, due to their uncertain and dynamic nature.

In this paper, we focus on security of SoSs: security is a non-compositional property so that, even if the individual constituent systems are secure, their interoperability may come together with new threats to the SoS security as a whole [3].

According to Ki-Arie et al. [7], there is yet "no clear guidance or limited tool-support integrating different modelling elements to visualize and assess the SoS security consequences". As surveyed in [8], testing provides a widely applied and practical means to assess a system security and many techniques have been proposed, but to the best of our knowledge there yet exists no specific approach for SoS security testing.

Summarizing, our research aims at developing an approach to assess security of SoS, named TeSSoS (Testing Security in System of Systems). In this paper, we lay the scene for such research, providing motivation and a plan of work, and a preliminary outline of the TeSSoS approach under development.

This paper is organized as follows: next section briefly revises related work for modeling SoS, managing security, and how security can be tested. Section 3 introduces a reference scenario on which an attack involving some systems working in a SoS is described. Section 4 frames our approach for assessing the security. Finally, Section 5 draws the conclusions.

II. Related work and Background

In the literature about SoSs architecture, we find four different SoS categorizations [9][10]: Directed, Acknowledged, Collaborative, and Virtual. The difference among such SoSs is how the constituent systems communicate and interact one with another to accomplish the common SoS purpose. In fact, the four architectures can be distinguished according to the answer to the following two questions: (1) Are there any central entity? (“*Yes”* for Directed and Acknowledged, and “*No”,* for the others) and (2) Do guidelines about the interaction among constituents exist? (“*Yes”* for Acknowledged and Collaborative, and “*No”* for the others).

The four different architectures are affected by different security issues depending on the responsibilities of system owners. In Directed and Acknowledged architectures, the main responsibility is from the central entity owner, which is responsible of coordinating the course of the activities. However, in Collaborative and Virtual architectures the responsible is not so well-defined. For Collaborative architecture it can be assumed that the responsible is the system that request for the joint work to accomplish the mission; on the other hand, in the case of Virtual each system is self-responsible, with the disadvantage of not having explicit collaboration from other systems side to support a common security policy. For the reference scenario, we are in this latter case because constituents are doing joint work without being conscious of this.

There exists work in progress for modeling these different SoS architectures based in Goal Oriented Requirement Engineering (GORE) [11], known as mKAOS [5]. This modeling language is an extension of KAOS/SySML [12], which is a language provided by the Object Management Group, (OMG).

In mKAOS each constituent system in a SoS is assigned a set of missions to accomplish, in the understanding that a mission is an objective that is carried out by two or more constituents. In this mKAOS language, the functionality of the whole system can be described as the sum of all joint functions of the systems. However, mKAOS does not support the representation of non-functional requirements so far. Non-functional requirements, and thus the representation of security requirements in the SoS, is still a pending work in the field. In particular, additional studies addressing security in the SoS according to its architecture and its validation through some test cases are needed.

To the best of our knowledge, there is not so much recent literature about security in SoS. Existing studies in SoS context are mainly related with risk evaluation [13] instead of the security requirements. Security is of course a topic largely addressed in isolated and distributed systems. These works outline some techniques that are used to manage the security in software system including models like CORAS, Attack Tree, Misuse cases, among others [14][15][16]. They help in early phases in representing the assets, vulnerabilities, weaknesses and attackers that the system may have. In this work we introduce an approach that aims to model security requirements for SoS for each SoS architecture.

Regarding to testing, according to its definition, is the dynamic verification of a program analyzing if the behavior is as expected on a finite set of test cases. Security testing techniques are used to guarantee that security mechanisms are implemented as expected, avoiding intended and unintended malicious behaviors. Security testing of a system focuses in testing security requirements and verifying system compliance with properties like authentication, authorization, availability, confidentiality, and non-repudiation. In our approach we include testing for the SoS covering the vulnerabilities that are not only from the specification of a single system, but of their joint work.

III. Reference scenario

Considering a real scenario, we would like to analyze the attack against Mat Honan [17]. This journalist had a personal blog, a twitter account, an amazon account as well as several email addresses, that we consider as constituent systems in his personal virtual SoS.

Honan had an enemy that wanted to delete all his tweets. Despite the attacker knew the Twitter username, Honan had a strong enough password. The attacker started to look for some relevant data that could help him in gaining access. At this time the attacker went to Honan’s personal blog and read the *whois* info. This disclosed some personal information like email address, postal address and phone number. Attacker tried then to go into this email account, supported by Gmail. However, once more, Honan’s password was strong enough. Gmail account used the reset password feature, that exposed another email address for recovery. This second email address was supported by iCloud. The attacker used again the recover-my-password function. This time the recovery method was not another recovery email, but to insert some digits of Honan’s credit card. At this juncture, the attacker had no more available sources of information. However, the attacker started thinking where they could find credit card data and ended by focusing Amazon. Recovering the password from Amazon website required to use an email address; this time they decided to use social engineering and made a phone call impersonating Mr. Honan. In a first call the attacker used the personal data they had got from *the whois* to impersonate Mr. Honan and asked for adding a new credit card to Honan’s Amazon account. The Amazon Service Desk did not notice anything strange and added this new and fake credit card. In a second call, using social engineering again, the attacker requested for password reset. To do so, the Service Desk asked to the pseudo Mr. Honan to say provide some digits of a credit card, and the attacker used the fake credit card they gave in the previous call.

At this point the attacker had enough to attack everything. They could enter Amazon and see Mr. Honan’s credit card number. Used this info to reset the password of iCloud and login. With the iCloud email they could then reset the Gmail password and once they were inside the Gmail inbox use the restore password of Twitter to delete every tweet.

What this true story teaches us is that, *despite every system in the Mr. Honan SoS was secure per se, the connection existing among these systems involved a set of insecurities that promoted the attack.*

IV. Contribution

The goal of our approach, named Testing for Security in System of Systems, (TeSSoS), is to design security requirements models in the target SoS and produce a set of test cases that help in evaluating the security of the SoS.

Security assessments are designed to measure an information system according to some criteria and seek potential security weaknesses. In this meaning, to assess the security of a SoS is to evaluate the security of the process that systems share when doing joint working to reach goals that these systems cannot reach by their own.

TeSSoS takes the attacker perspective to discover security flaws and propose new defensive features to developers. Common cyber-attacks have five stages [18]: *Reconnaissance*, *Scan*, *Gaining Access*, *Maintain Access* and *Clearing Tracks*.

An attack begins with the *reconnaissance*, i.e., the attacker gathering data about the SoS. This is important because the attacker needs to be sure that the risks involved with the attack are worthy for the benefits of perpetrating such attack.

Once the attacker has decided if attacking the company system is worthy enough, they move to *scan*. When the attacker starts scanning, they start discovering services, public IPs, and vulnerabilities, including social engineering. Attackers could consider even external services or systems, conforming their own SoS. When an attacker has designed an attack vector according to the vulnerabilities discovered, they can start to *gain access*. By exploiting the vulnerabilities found the attacker may successfully get into the SoS by trespassing a system through an existing vulnerability. After the attacker can access the system, they must *maintain access* by generating additional backdoors in case their initial accesses get patched and to avoid other attackers to kick them. Finally, the attacker begins *clearing tracks* to avoid get caught by forensics.

To avoid attacks to SoS succeed we introduce TeSSoS. The approach begins by discovering the SoS to be secured and helps analysts in finding threats and defining security features to enhance the security and test its correct development. The stages that comprises this approach are: *SoS Discovery*, *Red Requirements Specification*, *Blue Requirements Specification*, *Security Implementation*, and *SoS Evaluation and Validation*. In cyber-warfare, there are two teams: Red Team, responsible of identify vulnerabilities and find security holes, and Blue Team, responsible of finding and patching vulnerabilities. These names inspired us for the names of *Red Requirements Specification* and *Blue Requirements Specification*.

A. SoS Discovery

We start with *SoS Discovery* to discover what we are trying to secure. This stage, carried out by SoS analysts, focuses on producing a model that comprises the SoS. Since we need to discover each potential entry point an attacker could use, this is one of the most important stages. This SoS model helps in considering every constituent as well as the architecture that supports it. For each constituent we need to know what its operational capabilities and the handled data model are. Attacks could be also perpetrated against the database provider, our cloud service provider, and any digital provider on which we trust. To consider every perspective from an attacker, we must also consider third parties' systems and humans that are involved as soon as we are delegating on them our success, but also our failure. At this point, we have enough information about the architecture that supports the SoS.

The diagram for representing the SoS can be done by using mKAOS [5] diagram model that helps in knowing what we can consider the good and correct behavior. This is useful at the time of analyzing defenses because it allows us to observe if there is some strange or bad behavior as trace of an attack. The mKAOS model should be performed by an analyst jointly with a security expert.

Considering that SoS are geographically distributed and owned by different companies, for a full mKAOS model, collaboration among each constituent system owner is needed.

To balance the ratio of effort and cost to outcome, the assessment of the security for our assets starts by defining the scope. The scope must describe which attack are we defending from. It is needed to analyze what is the most dangerous threats for us, and thus develop most efficient and effective defenses for these ones. Once we have decided which are our most valuable assets and the most dangerous threats, we must determine potential attack vectors.

It is not always clear who is doing this. In Honan case which is a virtual SoS, he may be aware of potential attack and do by himself, or with guidance of an expert. The SoS of Mr. Honan could include which data is being shared, which systems are connected by any trace, what are the capabilities of each one, etc.

B. Red Requirements Specification

During the *Red Requirements Specification,* security analysts take the role of an attacker and try to find vulnerabilities by analyzing each constituent system and shared data among them. In those constituents for which we do not have additional information, we must consider it as a black-box and discover the vulnerability as a real attacker would do, just by analyzing the behavior of the systems when it is perturbed with some inputs [19].

The potential attacks found can be written in Gherkin language[1] which is used for having test cases that, additionally in our case, simulate these attacks. These attacks, designed by security or penetration testers, could be written in templates like these:

Feature: Attack action
Scenario: Using {*threat*} over {*vulnerability*}
GIVEN {*attacker*}finds{*vulnerability*} in {*asset*}
WHEN {*attacker*} uses {*threat*}
AND {*attacker*} is targeting {*vulnerability*}
THEN {*asset*} is exposed

A set of attacks written as features can be summarized into a *Red Product Backlog* that is used afterwards for designing *Blue Requirements Specifications*. This product backlog is useful for estimating the attack speed and eases the reading for design defensives features.

Some criteria to consider when prioritizing Red Requirements Specification could include the resources that attacker needs, what makes the SoS so important to be attacked, or how much effort is needed to succeed, among others. These criteria are important because an attacker will only use an attack in case they know their attack capabilities are higher than defensive ones, and thus they can succeed.

In Honan case, every constituent system is working as a black-box. Nevertheless, the attackers could still set up some attacks exploiting delegation of responsibility and sensitiveness of public data that could be subject to social engineering attacks, e.g. each system whose login is made through an email address is as secure as this email provider is; or having several systems that individually do not expose confidential data, when they are put together can reveal some sensible data.

C. Blue Requirements Specification

To prevent the attacks identified during the Red Requirements Specification, some countermeasures are designed during the *Blue Requirements Specification* stage, so the development team can implement them into the SoS.

One or more actions may be necessary to be protected against each attacker feature in the red product backlog. These defensive features written as User Stories [20] form the *Blue Product Backlog*. User Stories, written by SoS analysts and security analysts make it easier for developers and analyst of constituent systems to understand and implement those improvements into the constituents in the SoS. Honan's SoS is fully virtual, and he cannot modify the processes of these constituents. However, he could modify how these constituents are connected and reduce the amount of personal data that is publicly available to avoid being identified or tracked in case any constituent become attacked.

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

Defending user stories could be written in a template like this:

AS {*role*} **I WANT TO** {*defensive action*} **TO** {*protection against*}

As result, a product backlog with Defensor user stories is produced, summarizing the new security features to be implemented. At the time of producing this product backlog, it is important to give higher priority to user stories related to those vulnerabilities that expose the SoS the most, or those that could have a higher impact. This prioritization must be done by a security expert instead of development team that may not have such knowledge about security.

D. *Security Implementation*

In the stage of Security Implementation, different development teams of different SoS use the blue product backlog and encode new features that avoid attacks to be perpetrated successfully. Since new development features are written as user stories, the backlog can be used in agile development methodologies at any sprint. Updates can also be necessary to the operative systems, plugins, servers, etc. and the security and privacy policies may need to be changed to be more restrictive. Those blue user stories that apply to humans may consist on training courses instead of coding countermeasures.

E. *SoS evaluation and validation*

The last stage of TeSSoS is to validate if the security measures have been developed properly according to Blue Requirements Specifications. This validation must be carried out by security experts trying to have success in the attacks described in the Red Requirements Specifications using the Gherkin language. Notwithstanding, this validation must also consider the new security features that could have included new vulnerabilities not existing before in the SoS. Letting the security experts be creative and innovative as a real attacker would do may help in discovering new attack vectors that were not discovered at first sight.

If despite of the countermeasures, the security expert succeeds in the attack, our defenses would have failed, and it is needed to repeat the whole TeSSoS cycle again. Otherwise security experts could ensure that the system is robust enough to avoid the attacks that were considered. To continue being protected, TeSSoS could be relaunched considering a broader attack scope.

The evaluation in Honan's case would be to review a checklist to be sure that there is not a huge dependence from a single constituent to avoid an attack in cascade or confirm that no sensitive data are publicly available to prevent impersonation.

V. CONCLUSIONS

The contribution of this research is to provide an approach for assessing the security for SoS named TeSSoS. This security assessment takes the attacker perspective to discover security flaws and propose development of new features. SoS needs further research in this topic for providing mechanisms to support the security requirements in this context.

Consequently, we introduce TeSSoS, our security assessment approach. It is an ongoing work to assess the security of SoS. Security of the SoS can be analyzed by modeling security requirements from the attacker perspective, describing the security features to be later developed and generating relevant test cases. SoS security can be evaluated and validated by launching defined test cases that simulate real attacks.

In the future we will enhance the ideas presented in this paper by considering the security of SoS in a broader scope and continue TeSSoS development, considering more particularly the security testing, security features modelling, human factors relevance evaluation and control policies among others.

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] 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 CSUR, 48(2), 18, 2015.

[2] 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, Tucson, AZ, USA, 1993, pp. 238-244.

[3] D. J. Bodeau, "System-of-systems security engineering". In Proc. 10th ACSAC, 1994. (pp. 228-235). IEEE.

[4] E. Silva, E. Cavalcante, T. Batista, F. Oquendo, F. C. Delicato, P.F. Pires, "On the characterization of missions of systems-of-systems". In: Proceedings of the 2014 ECSA Workshops. ACM, 2014. p. 26.

[5] E. Silva, T. Batista, and F. Oquendo, "A mission-oriented approach for designing system-of-systems," in 2015 10th SoSE Conference, 2015, pp. 346–351.

[6] V. Chiprianov, K. Falkner, L. Gallon, and M. Munier. "Towards modelling and analysing non-functional properties of systems of systems". In 9th Int. Conference on SOSE, 2014 (pp. 289-294).

[7] D. Ki-Aries, S. Faily, H. Dogan, and C. Williams. "Assessing system of systems security risk and requirements with OASoSIS". In Proc. IEEE 5th International Workshop on ESPRE (pp. 14-20). IEEE. 2018

[8] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Security testing: A survey," Adv. Comput., vol. 101, no. March, pp. 1–51, 2016.

[9] W. G. J. Halfond, S. R. Choudhary, and A. Orso, "Penetration testing with improved input vector identification," Proc. - 2nd Int. Conf. Softw. Testing, Verif. Validation, ICST 2009, pp. 346–355, 2009.

[10] 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, 2008.

[11] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," Proc. Fifth IEEE Int. Symp. Requir. Eng., pp. 249–262, 2001.

[12] S. Tueno, R. Laleau, A. Mammar, & M. Frappier, "The SysML/KAOS domain modeling approach". arXiv preprint arXiv:1710.00903, 2017.

[13] 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.

[14] J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw, and N. R. Mead, "Requirements engineering for secure software". Addison-Wesley Professional, 2008.

[15] D. Meyer, J. Haase, M. Eckert, and B. Klauer, "A threat-model for building and home automation," Proc. IECON 2017 - 43rd Annu. Conf. IEEE Ind. Electron. Soc., vol. 2017–Janua, pp. 8126–8131, 2017.

[16] B. Solhaug and K. Stølen, "The CORAS Language – Why it is designed the way it is," Safety, Reliab. Risk Life-Cycle Perform. Struct. Infrastructures, pp. 3155–3162, 2013.

[17] How Apple and Amazon Security Flaws Led to My Epic Hacking {https://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/} Last visited: 16/Jan/2019

[18] K. Graves, CEH: official certified ethical hacker review guide. 2007.

[19] B. Beizer, "Black-box testing: techniques for functional testing of software and systems." 1995.

[20] M. Cohn, "User stories applied: For agile software development." Addison-Wesley Profession, 2004.