Available online at www.sciencedirect.com
ScienceDirect

Information Systems 32 (2007) 1166–1183

www.elsevier.com/locate/infosys

Security Attack Testing (SAT)—testing the security of information systems at design time[☆]

Haralambos Mouratidis[a,*], Paolo Giorgini[b]

[a]School of Computing and Technology, University of East London, UK
[b]Department of Information and Communication Technology, University of Trento, Italy

Received 7 June 2006; received in revised form 1 March 2007; accepted 15 March 2007
Recommended by: F. Carino Jr.

Abstract

For the last few years a considerable number of efforts have been devoted into integrating security issues into information systems development practices. This has led to a number of languages, methods, methodologies and techniques for considering security issues during the developmental stages of an information system. However, these approaches mainly focus on security requirements elicitation, analysis and design issues and neglect testing. This paper presents the Security Attack Testing (SAT) approach, a novel scenario-based approach that tests the security of an information system at the design time. The approach is illustrated with the aid of a real-life case study involving the development of a health and social care information system.

Keywords: Information systems development methodology; Integrating security and software engineering; Scenarios; Information system security testing

1. Introduction

Developers face many challenges in the development of modern information systems (ISs). These challenges are mainly associated with characteristics that modern ISs need to demonstrate, such as openness, adaptability, interoperability and security. Although many new techniques and technologies are being developed, on a regular basis, to enable developers to deal with most of these challenges, security has not yet received the attention it deserves.

Security engineering of ISs is mainly concerned with methods providing cost effective and operationally effective protection of ISs from undesirable events [1], and as Anderson claims [2], security engineering is about building systems to remain dependable in the face of malice, error or mischance. Current literature [3–6] extensively argues that in order to effectively design secure ISs, it is necessary to integrate security engineering principles into development techniques and introduce an ISs

[☆] This is an extended and substantially revised version of the "Using Security Attack Scenarios to Analyse Security During the Information Systems Design" paper presented in the 6th International Conference on Enterprise Information Systems (ICEIS), Porto – Portugal, ICEIS (3), 2004, pp. 10–17.

[*] Corresponding author. Tel.: +44 20 8223 3315.

E-mail addresses: haris@uel.ac.uk (H. Mouratidis), paolo.giorgini@unitn.it (P. Giorgini).

0306-4379/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2007.03.002

development methodology that will consider security as an integral part of the whole development process. However, there are various problems associated with the integration of security considerations during the development of ISs [4,7,8]:

1. A large number of ISs developers have very limited knowledge of security. However, in practice they need to develop ISs that require knowledge of security.
2. There are many security-related concepts and definitions that are used differently by security specialists and ISs developers. As a result, there is an abstraction gap that makes the integration of security into information system development practices more difficult.
3. It is difficult to define together security and functional components and at the same time provide a clear distinction. For instance, which components are part of the security architecture and which ones are part of the functional specification.
4. It is difficult to move from a set of security requirements to a design that satisfies these requirements, and also understand what are the consequences of adopting specific design solutions for such requirements.
5. It is difficult to get empirical evidence of security issues during the design stages. This makes the process of analysing security during the design stage more difficult.
6. It is difficult to test the proposed security solutions at the design level.

All these problems raise a number of research questions and challenges: *How developers with minimum knowledge of security can develop secure ISs? What are the requirements for structured methods and methodologies to support security analysis during the development process? Is it possible to define security requirements together with functional requirements and at the same time provide a clear distinction indicating which are the security requirements? Is it possible to test the developed solution with respect to security at design time?*

Although the current state of the art fails to provide solutions and answers to all the above problems and research challenges, in the last few years a considerable number of promising works have appeared. A number of ontologies and modelling languages [9–12] have developed aiming to narrow the gap and create a common ground for the integration of security issues into ISs development processes. Methodologies and methods [4,8,13] are under development aiming to provide a structured approach towards the integration of security issues in the development process and allow simultaneous definition of security and functional requirements. We have also contributed to this line of research. In previous work we have proposed secure Tropos [4,14,15], a methodology that considers security issues as part of the ISs development process by employing the same concepts and notations throughout the development process. We then enhanced the secure Tropos methodology by providing a set of security patterns [16] to assist developers with limited knowledge of security to produce a security-aware design. However, neither secure Tropos nor any of the existing approaches have focused on providing a process to test at design time the security solution that derives from the application of a structured ISs development methodology. Such a testing process does not aim to substitute the existing security testing techniques [6,17,18], but rather to compliment them and in effect act as an initial filter to identify at design time security problems of the developed system and allow developers to redefine the system and improve its security at an early stage of the development. It is well known that problems identified during the design stage are easier and less expensive to fix than problems identified in subsequent stages of the development process.

In this paper we introduce the Security Attack Testing (SAT) process; a novel scenario-based approach that assists developers to test, at design time, the developed system against potential security attacks. Our approach is based on five key ideas:

- employ the same concepts in testing as in the requirements elicitation, analysis and design;
- test during the design time;
- employ scenarios to test the security of the system;
- create security scenarios and derive the test cases through a systematic process;
- integrate the testing approach to the secure Tropos development process.

To show the applicability of our approach we revisit the electronic single assessment process system case study [19]; the analysis and design of which has been presented in the literature [14,15] and we illustrate how the proposed approach can be used to test and improve the security of the system.

Section 2 of the paper provides a brief review of secure Tropos, whereas Section 3 discusses security testing. Section 4 describes the proposed Security Attack Testing (SAT) process, and Section 5 discusses related work. Section 6 concludes the paper and points out directions for future work.

2. Secure tropos

This paper is not intended to provide a full description of the analysis and design stages of the secure Tropos methodology. Such descriptions are widely available in the literature [4,14,15,20]. However, it is important to provide a brief introduction to the secure Tropos methodology to enable readers not familiar with it to understand the fundamentals of it, so they can easily understand the rest of the paper.

Secure Tropos is based on the Tropos methodology [21], which uses the concepts of actor (entity that has strategic goals and intentionality), goal (an actor's strategic interest), soft-goal (goal without clear criteria whether it is satisfied or not), task (it represents the way of doing something), resource (it represents a physical or informational entity, without intentionality) and social dependencies (indicate that one actor depends on another in order to attain some goals, execute some tasks, or deliver a resource).

Secure Tropos extends the Tropos methodology by adding security concerns during the development of ISs. In particular, secure Tropos extends the Tropos language as well as its development process. The language extension consists of redefining existing concepts with security in mind as well as introducing new concepts. A *security constraint* is defined as a restriction related to security issues, such as privacy, integrity and availability, which can influence the analysis and design of the information system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives [22]. Secure Tropos uses the term *secure entity* to describe any goals and tasks related to the security of the system. A *secure goal* represents the strategic interests of an actor with respect to security. Secure goals are mainly introduced in order to achieve security constraints that are imposed on an actor or exist in the system. However, a secure goal does not particularly define how the security constraints can be achieved, since alternatives can be considered [22]. The precise definition of how the secure goal can be achieved is given by a secure task. A *secure task* is defined as a task that represents a particular way for satisfying a secure goal. *A secure dependency* introduces security constraint(s) that must be fulfilled for the dependency to be satisfied. Both the depender and the dependee must agree for the fulfilment of the security constraint in order for the secure dependency to be valid. That means the depender expects from the dependee to satisfy the security constraint(s) and also that the dependee will make an effort to deliver the dependum by satisfying the security constraint(s).

Fig. 1 provides a graphical representation of the above concepts.

Based on the above concepts, the process in secure Tropos is one of analysing the security needs of the stakeholders and the system in terms of security constraints imposed on the stakeholders and the system, identifying secure entities that guarantee the satisfaction of the security constraints, and assigning capabilities to the system to help towards the satisfaction of the secure entities. In particular, as for Tropos, the secure Tropos methodology covers four main phases [15]:

- During the *early requirements analysis* phase the security reference diagram [14,22] is constructed and security constraints are imposed on the stakeholders of the system (by other stakeholders). During this stage, imposed security constraints are expressed, initially as high-level

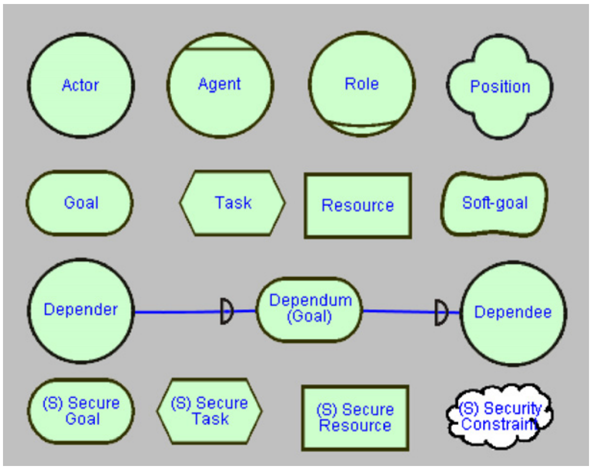


Fig. 1. Tropos and Secure Tropos notation.

statements which are later further analysed. Then secure goals and entities are introduced to the corresponding actors to satisfy the security constraints.
- During the *late requirements analysis* phase, security constraints are imposed on the system-to-be (by reference to the security reference diagram). These constraints are further analysed according to the analysis techniques [15,22] of secure Tropos and security goals and entities necessary for the system to guarantee the security constraints are identified.
- During the *architectural design* any possible security constraints and secure entities that new actors might introduce are analysed. Additionally, the architectural style of the information system is defined with respect to the system's security requirements and the requirements are transformed into a design with the aid of security patterns. Furthermore, the agents of the system are identified along with their secure capabilities.
- During the *detailed design* phase, the components identified in the previous development stages are designed with the aid of Agent Unified Modeling Language (AUML). In particular, agent capabilities and interactions taking into account the security aspects are specified with the aid of AUML. The important consideration, from the security point of view, at this stage is to specify the components by taking into account their secure capabilities. This is possible by adopting AUML notation.

3. Security testing

Security testing is widely considered an important activity that helps to identify security vulnerabilities. Existing testing techniques, such as network mapping and vulnerability scanning [6,17], have been used for many years and automated security analysis tools, such as Nessus and Retina, are considered valuable solutions. However, such techniques and tools only work for systems that are already built, i.e. they are useful after the system has been developed. As development methodologies start to consider security issues throughout the development process, it is important to test the design models, rather than just the implementation, to ensure that the design of the system enforces the necessary security requirements. However, as discussed in the introduction, current state of the art fails to provide structured and well-defined processes to test at the design time the security of an information system. Therefore, a security testing process is needed. Such process should demonstrate at least the following characteristics:

- It should be clear and well guided. The concepts used in the process must be well defined and the activities and steps of the process well explained.
- It should be flexible enough to allow integration into a methodological framework. The guidelines and the structural processes of the methodology will allow the explicit definition of the applicability of the security testing process within the stages of the methodology.

To meet the above requirements, we have chosen a scenario-based testing method. Our decision is based on the fact that scenarios can be represented in various ways [23]. This allows better adoption to a methodology's concepts and notations and therefore better integration within a methodology's development lifecycle. Moreover, the effectiveness of scenario-based methods has been widely tested in many different areas of computer science research, such as software engineering [24], business-process reengineering [25], and user interface design [26] and for different activities such as eliciting information about a system's requirements, communicating with stakeholders, providing context for requirements [23] and validation of requirements [23,27].

4. A security attack testing (SAT) process

Our scenario-based security testing process aims to test, at the design time, the system's security against potential attacks. In doing so, two sets of scenarios—*dependency* and *security attack*—are identified and constructed and *Security Test Cases* are defined, from the scenarios, to test the developed design of the system against various types of attacks.

In particular, SAT aims to identify the goals and the intentions of possible attackers; identify through these a set of possible scenario attacks to the system; and apply these attacks to the system to see how it copes. By analysing the goals and the intentions of the attackers, the developer obtains valuable information that helps to understand not only the *how* the attacker might attack the system, but also the *why* an attacker wants to attack the system. This leads to a better understanding on how possible attacks can be prevented. In addition, the application of a set of attacks to the system contributes towards the identification of attacks that the system might not be able to cope and this leads to the redefinition of the agents of the system and the addition of new secure capabilities to the system to assist in the protection of these attacks.

As discussed earlier, one of our objectives was to integrate the scenario-based security testing approach into the development process of the secure Tropos methodology. However, this was not an easy task and various research questions and challenges surfaced: *In which stage of the development process should the approach be integrated? What are the implications of such integration for the rest of the development process? What kind of inputs the proposed approach receives from earlier stages and how it affects consequent stages?*

Our initial analysis concluded that *Early* and *Late* requirement phases did not provide enough information with respect to the architecture of the system which is necessary for the construction of the security scenarios. To identify the scenarios required by our approach, information is needed related to the agents and resources of the system as well as the communication paths and the secure capabilities of each of the agents of the system.

Both *Architectural* and *Detailed design* phases were candidates for integrating our proposed approach. The main advantage of the detailed design is that the components of the system have been defined in more detail, as opposed to the more abstract definition of components during the architectural design stage. On the other hand, the main advantage of the architectural design is that if redefinition of the system is needed, this could take place faster and with fewer expenses. Therefore, this leads to a situation where a trade-off is required between testing the system earlier and producing some extra test cases. However, the factors influencing such decision differ for different projects.

Therefore, we decided to develop our scenario-based testing approach in such a way that a pragmatic solution can be adopted for its integration to the development stages of the secure Tropos methodology. That is, the approach can be employed if necessary throughout both architectural and detailed design. For instance, the dependency scenarios and a number of security attack scenarios can be constructed during the architectural design stage, whereas an extra number of security attack scenarios together with the test cases can be defined during the detailed design. This provides flexibility to the developers. However, experience on applying the approach [22] indicates that most of the times, it will be employed during the latter part of the architectural design.

Fig. 2 illustrates where our Security Attack Testing process stands in the secure Tropos process. The SAT process receives inputs mainly from the architectural design stage for creating the initial scenarios. In cases where limitations are identified in the security of the system, after running the test cases, and new components need to be added, this information is fed back to the architectural design where the new components are added into the system's architecture and then the SAT process runs again. Moreover, if information is required from the detailed design stage, such as detailed description of a particular component for the construction of a scenario or a test case, this information is input into the SAT process. When the test cases have successfully run and no security limitations are identified,

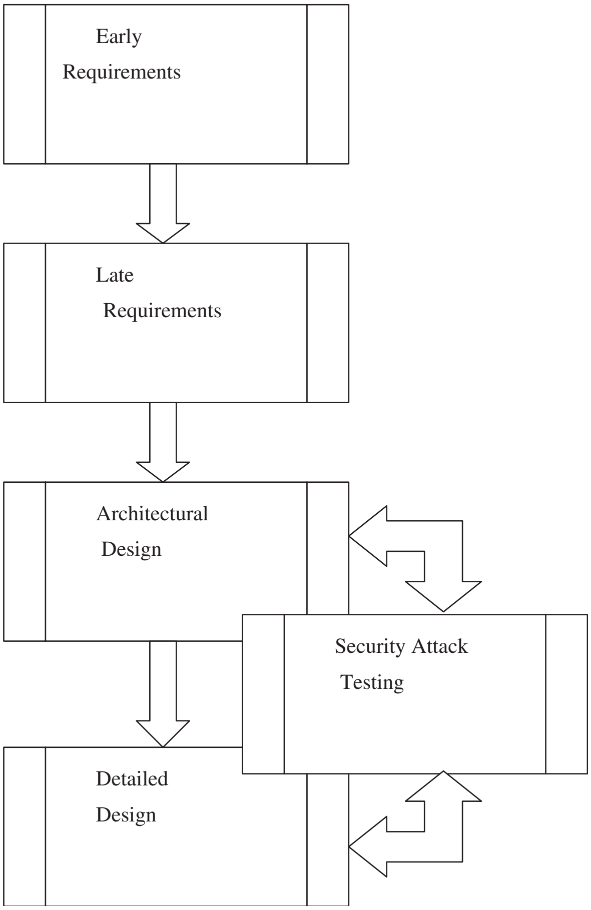


Fig. 2. The secure Tropos stages and SAT.

the results of the SAT process are fed into the detailed design phase of the secure Tropos methodology.

The SAT process includes four main activities: *Derive Dependency Scenarios*, *Define Security Attack Scenarios*, *Define Security Test Cases* and *Redefine the System*. Each of these activities includes a number of sub-activities as shown in Fig. 3.

In theory a developer should follow a sequential process when employing the above activities. The dependency scenarios should be defined before the security attack scenarios since information from the former activity is used as input for the latter activity. Similarly, the security attack scenarios should be defined before the Security Test Cases are defined. However, in reality the process is highly iterative and usually more than one iterations are required where the findings of one (sub) activity might feed back a previous (chronologically) activity as indicated with the arrows pointing from a later to an earlier activity in Fig. 3. Especially, this is the case with the last activity *Redefine the system*. If during the *Define Security Test Cases* activity is concluded that the system cannot defend against some of the attacks, the results of the failing test cases are fed back to the previous activities and the process starts again. Depending on the extent of the problem, the results can be fed to any of the previous activities. For instance if new components of the system need to be defined, the new components must be added to the system and the testing process will go back to the first activity where new dependency scenarios will be defined taking into account the new components.

The following section describes each one of the above activities in detail. To support the description of these activities and to demonstrate their usefulness and applicability we revisit the electronic single assessment process system case study which was used to initially motivate the development of the secure Tropos approach [15,22]. The electronic Single assessment Process system is a health and social care information system to support the effective care of older people.

Therefore, as part of the description of each activity of our approach, we employ the eSAP case study and we demonstrate how our approach can be used to test the security of the eSAP system and how it actually identifies limitations and improves the security of the system.

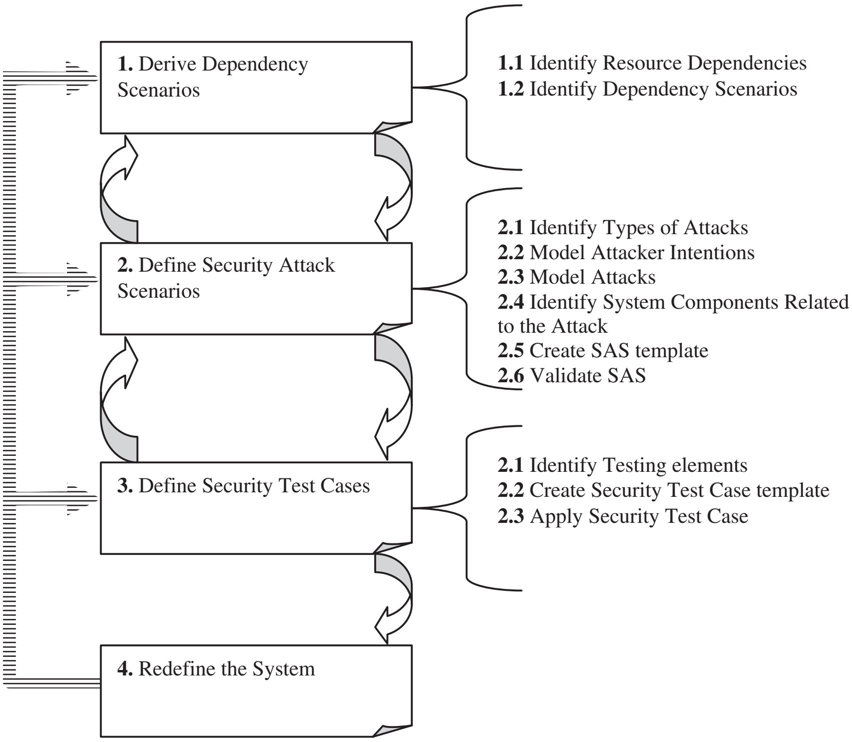

1. Derive Dependency Scenarios
1.1 Identify Resource Dependencies
1.2 Identify Dependency Scenarios
2. Define Security Attack Scenarios
2.1 Identify Types of Attacks
2.2 Model Attacker Intentions
2.3 Model Attacks
2.4 Identify System Components Related to the Attack
2.5 Create SAS template
2.6 Validate SAS
3. Define Security Test Cases
2.1 Identify Testing elements
2.2 Create Security Test Case template
2.3 Apply Security Test Case
4. Redefine the System

Fig. 3. SAT Activities.

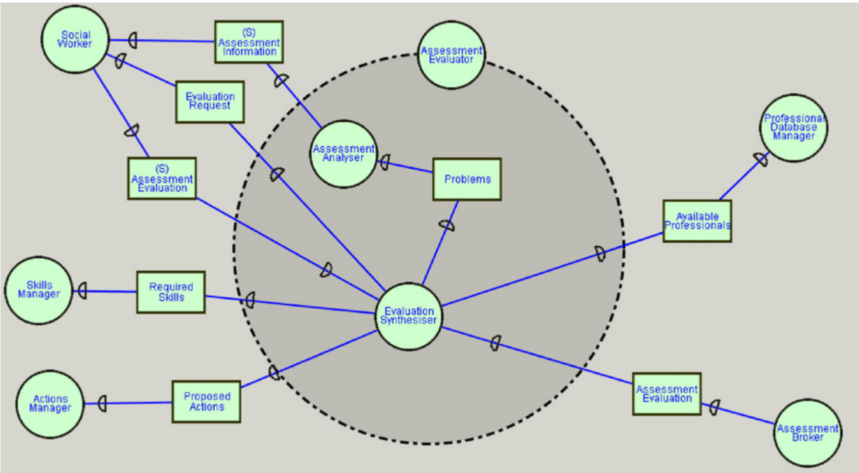


Fig. 4. Extended actors diagram for Assessment Evaluator.

4.1. *Derive dependency scenarios*

The aim of this activity is to identify scenarios that involve actors and resources of the system-to-be. By identifying such scenarios, developers identify areas of the system where potential attacks might take place. The output of this activity is used as an input for the definition of the security attack scenarios.

4.1.1. *Identify resource dependencies*

This sub-activity involves the identification of resource dependencies and related actors of the system under development. Such dependencies can be identified by examining extended actor diagrams,[1] which were constructed during the latest phases of the architectural design stage of secure Tropos. In particular, every resource dependency modelled in an extended actor diagram is identified and it is used as the starting point for the definition of a dependency scenario.

Consider for instance the eSAP case study. Fig. 4 illustrates an extended actor diagram with respect to the Assessment Evaluator actor as derived from the early and late requirements analysis of the eSAP case study [22].

Various resource dependencies can be identified from that extended actor diagram. For instance, the Social Worker depends on the Evaluation Synthesiser for the Assessment Evaluation secure dependency. In turn, the Assessment Analyser depends on the Social Worker for the assessment information secure dependency.

4.1.2. *Identify dependency scenarios*

A dependency scenario is identified by taking into account all the resource dependencies between the same actors. For each identified scenario, a textual description is generated to complement the graphical notation of secure Tropos and convey the scenario easier.

Consider again for instance the eSAP case study. By considering all the resource dependencies between the same two actors, a dependency scenario is derived. For example, from the extended actors diagram presented in Fig. 4, we identify the following scenario for the Social Worker and the Assessment Evaluator actors:

A Social Worker depends on the Assessment Evaluator to obtain an assessment evaluation. For this reason, the Social Worker sends an evaluation request to the assessment evaluator along with the assessment information. Then, the Assessment Evaluator returns the Assessment Evaluation.

4.2. *Define security attack scenarios*

Before explaining the second activity of the SAT process, it is important to discuss some definitions and concepts related to it.

The definition of attack that we use is one proposed by Matt Bishop [17], according to which an *attack* is an action that might cause a potential violation of

[1] An extended actor diagram captures the actors of the system identified during the architectural design together with any dependencies they might have with existing actors of the system. For more information please see [21].

security in the system. The person, or software, who executes such action is called an *attacker*.

A *Security Attack Scenario* (SAS) is defined as an attack situation describing the actors of an information system and their secure capabilities as well as possible attackers and their goals. It identifies how the secure capabilities of the system prevent (if they prevent) the satisfaction of the attackers' goals.

A security attack scenario involves possible attacks to an information system, a possible attacker, the resources that are attacked, and the actors of the system related to the attack together with their secure capabilities. Security Attack Scenarios are modelled as enhanced secure Tropos actor diagrams. In particular, an attacker is depicted as an actor who aims to break the security of the system. The attacker intentions are modelled as goals and tasks and their analysis follows the same reasoning techniques that the Tropos methodology employs for goal and task analysis. For the purpose of a security attack scenario, a differentiation takes place between internal and external actors of the system. *Internal actors* represent the core actors of the system whereas *external actors* represent actors that interact with the system. Such a differentiation is essential since it allows developers to identify different attacks to resources of the system that are exchanged between external and internal actors of the system.

For each dependency scenario identified during the previous activity, one or more security attack scenarios are defined. There are six sub-activities in defining a security attack scenario: *Identify Types of Attacks*; *Model Attacker Intentions*; *Model Attacks*; *Identify System Components Related to the Attack*; *Create SAS template*; and *Validate SAS*. These are described below.

4.2.1. Identify types of attacks

The security requirements and goals identified during the early requirements analysis provide the input for this activity. In particular, the first step involves the identification of the different types of attacks. The first step in identifying security attacks is to consider the security requirements of the system and to identify potential categories of attacks that might endanger these requirements. In doing so, libraries of attacks, attack trees and attack patterns can be employed to provide a comprehensive list of possible attacks. The activity concludes with the creation of a list of the different types of attacks together with a brief description.

As an example, consider the eSAP case study. As derived from the analysis of the eSAP system [22], the two main security features are privacy and integrity. According to Stallings [28], the following categories of attacks can be identified that can endanger the above security features.

1. *Interception Attack*, in which an unauthorised party, such as a person, a program or a computer, gains access to an asset. This is an attack on privacy.
2. *Modification Attack*, in which an unauthorised party not only gains party to but also tampers with an asset. This is an attack on integrity.

4.2.2. Model attacker intentions

When the types of attacks have been identified, the intentions of an attacker according to these types of attack are modelled. Tropos goal diagram notation is used for analysing the intentions of an attacker in terms of goals and tasks. The root goal of the attacker derives from the type of attack considered. Then this goal is decomposed to subgoals and tasks that capture more precise the intentions of the attacker. Some of these goals/tasks can be identified by considering the threats (related to the type of attack) considered on the security reference modelling [22] activity during the early requirements development stage. However, other goals could be derived from the analysis of a possible attacker's intentions following goal-reasoning techniques [21]. This is due to the fact that an attack is an exploitation of a system's vulnerability, whereas a threat is a circumstance that has the potential to cause loss or harm [29]. Therefore, an attack can lead to a threat only if the exploitation of the vulnerability leads to a threat. This means that some attacks can be successful but do not lead to threats as other system features protect the system. As with the previous step, attack trees and libraries might also be employed to identify possible goals of the attacker. This activity ends with the construction of a goal diagram of an attacker together with their intentions in terms of goals and tasks.

As an example, consider again the eSAP case study and let us assume an interception attack scenario, where a possible attacker aims to attack the privacy of the system in order to obtain information such as private assessment data or private care plan information. This is the root goal of the attacker. Possible goals (amongst others) towards the satisfaction of the root goal are *read the data* and *get access to the system*. According to the security reference diagram analysis [22], tasks to accomplish these goals are social engineering, password sniffing and eavesdropping. Such analysis is modelled as illustrated in Fig. 5.

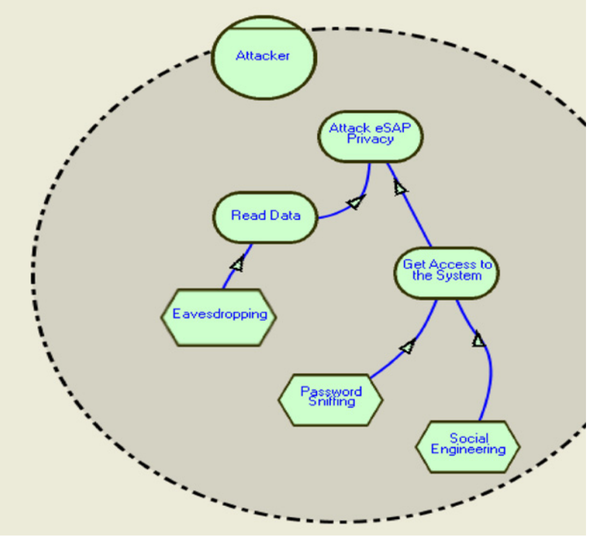


Fig. 5. Example of attacker modelling.

4.2.3. *Model attacks*

When the intentions of the attacker have been considered, the next activity aims to model the attacks to the system that result from the identified intentions. For each dependency scenario identified during the previous activity, potential attacks are modelled. Attacks are depicted as dash-lined links, called *attack links*, which contain an "attacks" tag. The attack links initiate from one of the attacker's tasks and end at the attacked resource. The activity concludes with a goal diagram of the attacker together with the possible attacks to the identified dependency scenarios. An example of such a diagram for the eSAP case study is shown in Fig. 6. In this diagram various attacks are modelled. For instance, password sniffing attacks endanger the system access clearance resource dependency whereas eavesdropping attacks endanger the system access request dependency.

4.2.4. *Identify system components related to the attack*

The next activity aims to identify and model the agents (internal and external) of the system related to the identified attack(s). The secure capabilities, of each agent, that help to prevent the attacks are identified and dashed-links (with the tag "help") are provided indicating the capability and the attack they help to prevent. The result of this activity is an actor diagram modelling internal/external agents of the system together with their secure capabilities. A link that carries the "help" tag is used to indicate the attacks these actors help to prevent. For instance, for the eSAP case study various system components are related to the interception attack modelled in the previous activities, such as the eSAP Guard; the Authenticator; and the Cryptography Manager. Moreover, as shown in Fig. 7 the Social Worker actor has also secure capabilities that help towards the prevention of some of the attacks.

4.2.5. *Create SAS template*

The last sub-activity is the creation of the security attack scenario template. Information from the previous activities is gathered and the security attack scenario template is used for each of the scenarios that have been identified. The template includes eight fields: *SAS ID*—this is a unique number that identifies the Security Attack Scenario modelled; *SAS Name*—this is a unique name that identifies the Security Attack Scenario modelled; *Author*(*s*)—the name(s) of the author(s) of the SAS; *Attack Type*—the type of attack that this scenario corresponds. The attack type should match one of the types identified during the first activity of the scenario creation process; *System Actors Involved*—the actors (internal and external) of the system involved on the scenario. These actors have been identified during sub-activity in Section 4.2.4; *Scenario Trigger*—the situation that triggers the scenario. This is effectively the dependency scenario for which the attack scenario corresponds to; *Textual Description*—a textual description of the scenario focusing on the analysis of the attacker and the resources under attack; *Graphical Representation*—a goal diagram of the security attack scenario. This goal diagram includes the diagrams modelled during sub-activities in Sections 4.2.2–4.2.4. A simplified[2] template for the interception scenario of the eSAP system as results from the

[2] Not all the goals of the attacker are illustrated in this SAS. We have kept the goals of the attacker to minimum number to allow an easier understanding of the approach. For more attack goals please see [14].

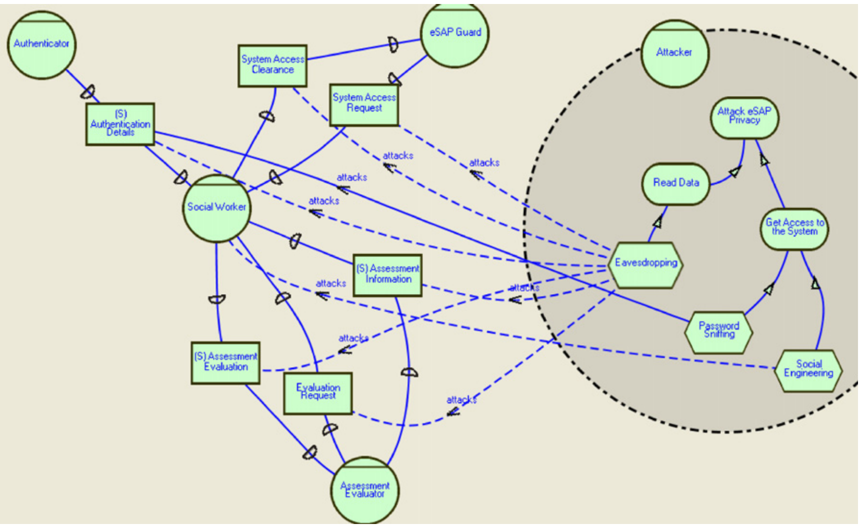


Fig. 6. An example of modelling attacks.

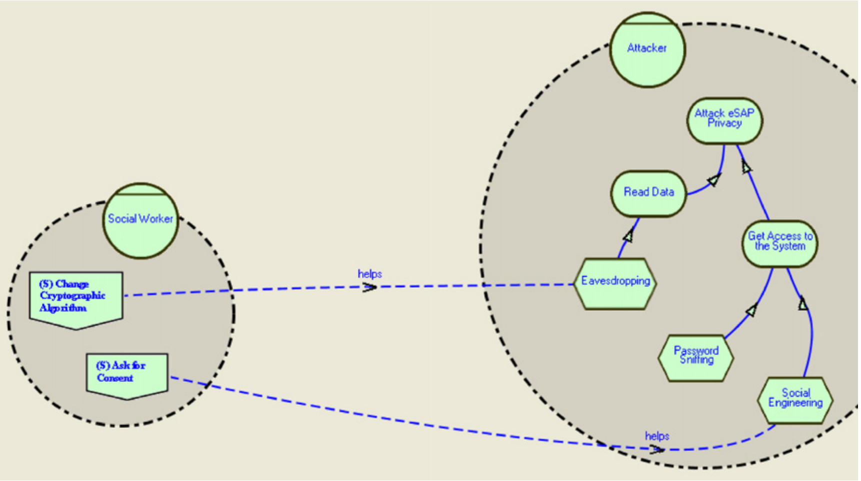


Fig. 7. Example of "help" capabilities.

analysis presented in the previous activities is shown in Fig. 8.

4.2.6. Validate SAS

When the security attack scenarios have been created the next step involves their validation. Software Inspections are used for the validation process. Software inspections have been proved as effective means for document-based validation [30] and in fact experiments [31] have demonstrated that in some cases software inspections are more effective than extensive testing. Moreover, as Sommerville [32] indicates "*software inspections do not require the program to be executed*". This fits perfectly to our purpose, which is to validate our scenario on the design level without necessarily having a code corresponding to these scenarios.

For our project, the inspection of the scenarios involves the use of validation checklists. Although the information, questions and length of such checklists might vary, there are some attributes that all of them should demonstrate to be considered satisfactory: syntax, completeness and consistency. Syntax-related items of the list aim to identify any possible violations of the secure Tropos and the scenarios' syntax.

The completeness related items of the list aim to check the completeness of the developed security

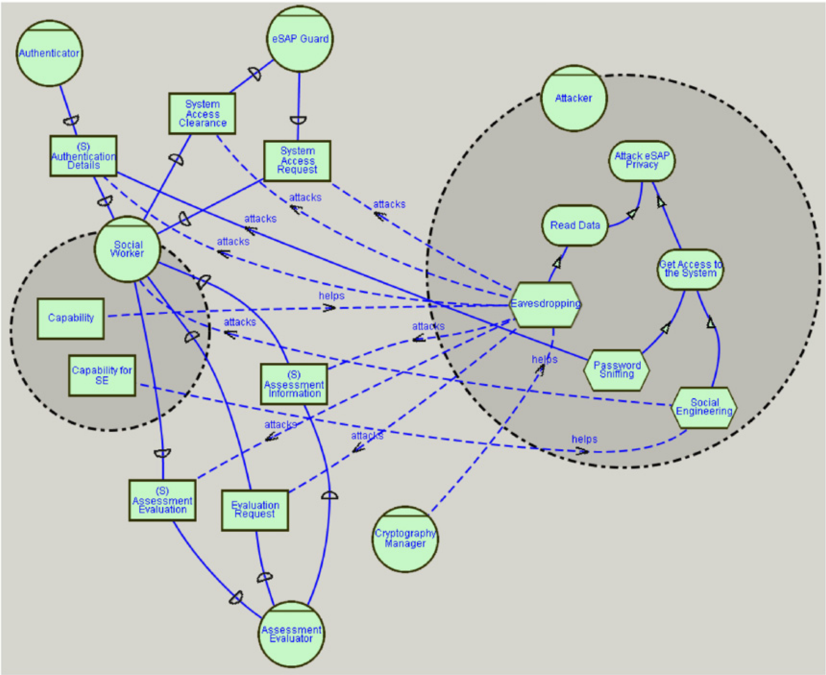
<table>
<tr><td><b>SAS ID</b></td><td><b>01</b></td><td><b>SAS Name</b></td><td><b>Interception</b></td></tr>
<tr><td><b>Authors</b></td><td><b>H. Mouratidis</b></td><td><b>Attack Type</b></td><td><b>Privacy</b></td></tr>
<tr><td><b>System Actors Involved</b></td><td colspan="3"><b>eSAP Guard, Social Worker, Authenticator, Assessment Evaluator, Cryptography Manager</b></td></tr>
<tr><td><b>Scenario Trigger</b></td><td colspan="3"><b>The Social Worker actor exchanges data with the Assessment Evaluator.</b></td></tr>
<tr><td colspan="4"><b>Textual Description</b></td></tr>
<tr><td colspan="4">The attacker's main goal can be decomposed to Read Data and Get Access to the System sub-goals. The first sub-goal involves the attacker trying to read the data that it is transmitted to and from the eSAP system, whereas the second sub-goal involves the attacker trying to break into the system and gain access to it. According to the eSAP specification [23] to accomplish the first sub-goal the Attacker should try to read the data transferred between the Social Worker and the eSAP system's actors such as the Assessment Evaluator and the Authenticator. To accomplish the second sub-goal, the Attacker might use password sniffing or social engineering. In the first case, the Attacker scans all the resources that flow in the network looking for passwords whereas in the case of social engineering, the Attacker tries to deceive the Social Worker in order to obtain valuable information, such as their authorisation details that will allow them to gain access to the system.</td></tr>
<tr><td colspan="4"><b>Graphical Representation</b></td></tr>
<tr><td colspan="4">
</td></tr>
</table>

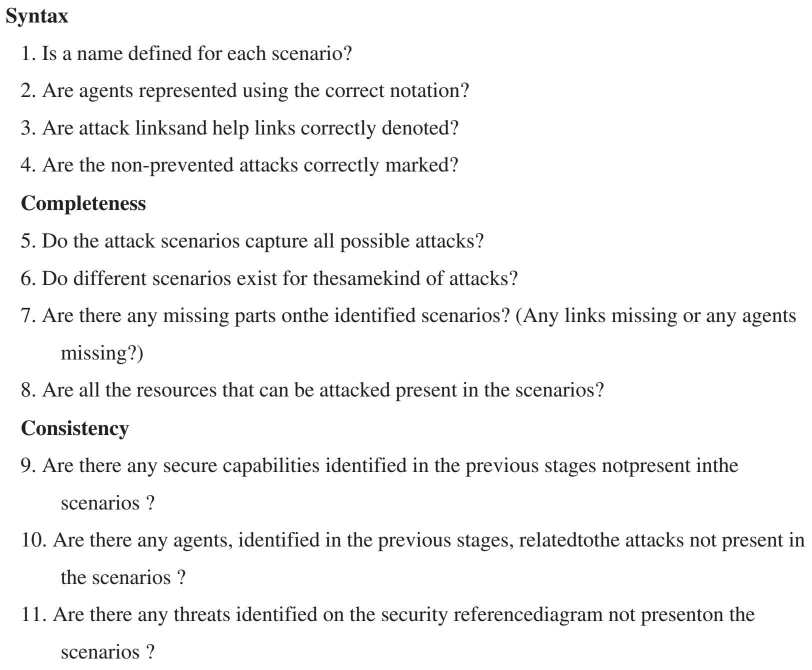
Fig. 8. Interception attack template.

attack scenarios. The consistency-related items of the list aim to identify any possible inconsistencies within the security attack scenarios but also between the security attacks scenarios and the secure Tropos models developed during the previous stages of the development process. An example of the validation checklists used to validate the Security Attack Scenarios for the eSAP case study is shown in Fig. 9.

Although inspections have been proposed by this research for the validation of the security attack scenarios; other techniques could also be applied depending on the developers' experience and the nature of the system. For instance, two well-known validation techniques for requirements specification are walkthroughs and prototyping [30].

4.3. *Define security test cases*

When the scenarios have been validated, the next activity aims to identify test cases and test, using these test cases, the security of the system against



Syntax

1. Is a name defined for each scenario?
2. Are agents represented using the correct notation?
3. Are attack linksand help links correctly denoted?
4. Are the non-prevented attacks correctly marked?

Completeness

5. Do the attack scenarios capture all possible attacks?
6. Do different scenarios exist for thesamekind of attacks?
7. Are there any missing parts onthe identified scenarios? (Any links missing or any agents missing?)
8. Are all the resources that can be attacked present in the scenarios?

Consistency

9. Are there any secure capabilities identified in the previous stages notpresent inthe scenarios ?
10. Are there any agents, identified in the previous stages, relatedtothe attacks not present in the scenarios ?
11. Are there any threats identified on the security referencediagram not presenton the scenarios ?

Fig. 9. Example of validation checklist.

any potential attacks. According to the IEEE Standard 610 [33] a test case is defined as "*a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement*". A more recent definition by Ron Patton [34, p. 65] indicates that "*test cases are specific inputs that you will try and the procedures that you will follow when you test the software*". Our understanding of test cases is consistent with these definitions and in our work each test case is derived from a possible attack depicted in the security attack scenarios.

4.3.1. *Identify testing elements*

Each Security Attack Scenario might contain more than one attack. It is important for testing purposes to produce different test cases for each of the attacks. In doing so, the first activity involves the isolation of a particular attack and the identification of the testing elements related to this attack. The testing elements are components of the security attack scenarios which are used as inputs for the test case generation and execution. In particular, the resource under attack, and the system defences against the attack (as derived from the related to the attack actors and their secure capabilities) constitute the testing elements.

For instance, for the eSAP case study, various testing elements can be identified from the interception security attack scenario. These include (amongst others) related resources, such as the assessment information and the assessment evaluation; secure capabilities, such as ask for consent and change cryptographic algorithms; and related actors such as the Authenticator, the eSAP Guard, and the Cryptography Manager.

4.3.2. *Create security test case*

Although our approach does not restrict the developers for the generation of the test cases, it is important to highlight some guidelines to be followed to ensure that test cases are easy to read and that all the security-related aspects of the system are adequately tested:

- For each attack-related task of an attacker, identified during the scenario creation stage, a test case should be generated.
- The discussion of a test case should take into account the testing elements identified during the previous sub-activity. In particular, all the secure capabilities of the agents of the system which

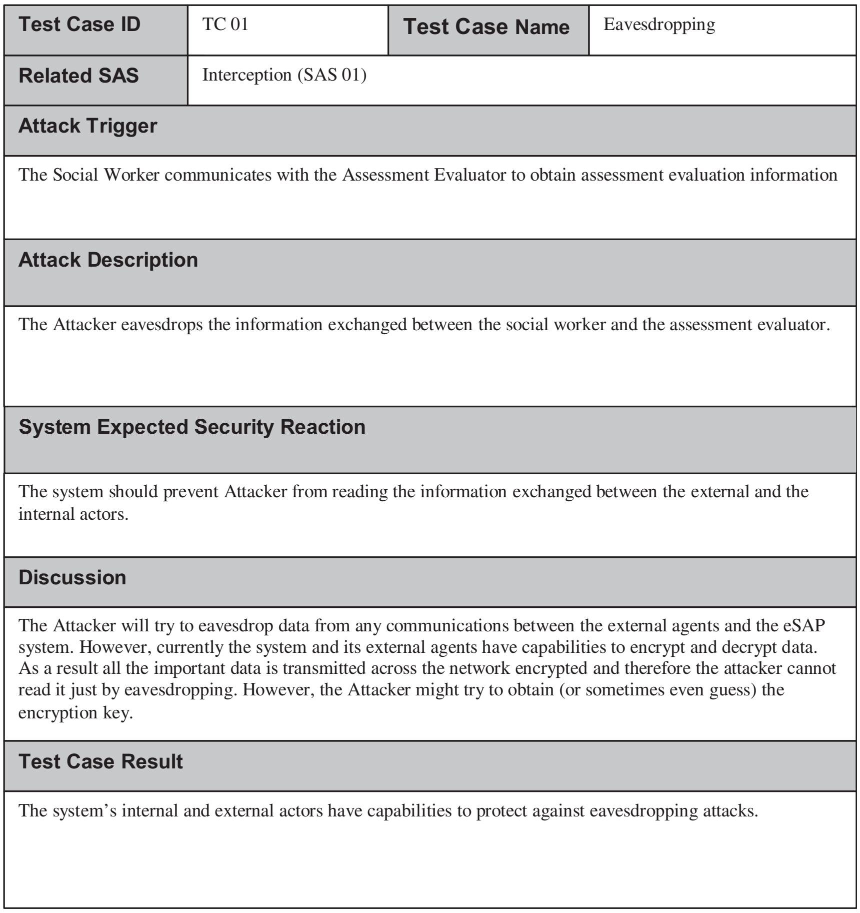
contribute towards the defence of the attack must be considered.

- Each test case is documented with the aid of a Security Test Case template. This template includes eight fields: *Test Case ID* which indicates the unique ID of the Test case; *Test Case Name* which indicates the name of the test case; *Related SAS* which indicates the security attack scenarios related to the test case; *Attack Trigger* which indicates the condition that triggers the attack; *Attack Description* which provides a textual description of the attack; *System Expected Security Reaction* which indicates in a textual format the expected reaction of the system to the attack; *Discussion* which provides an in-depth discussion of the attack and the system defences, and it forms the basis for the result of the test case; *Test Case Result* which indicates the output of the test case and it suggests possible improvements. It is worth mentioning that the last field of the template is filled in during the next activity.

Example of test cases related to the eSAP interception scenario are shown in Figs. 10–12. For instance, Fig. 10 models an eavesdropping attack which is triggered when the social worker communicates with the assessment evaluator in order to obtain assessment related information.

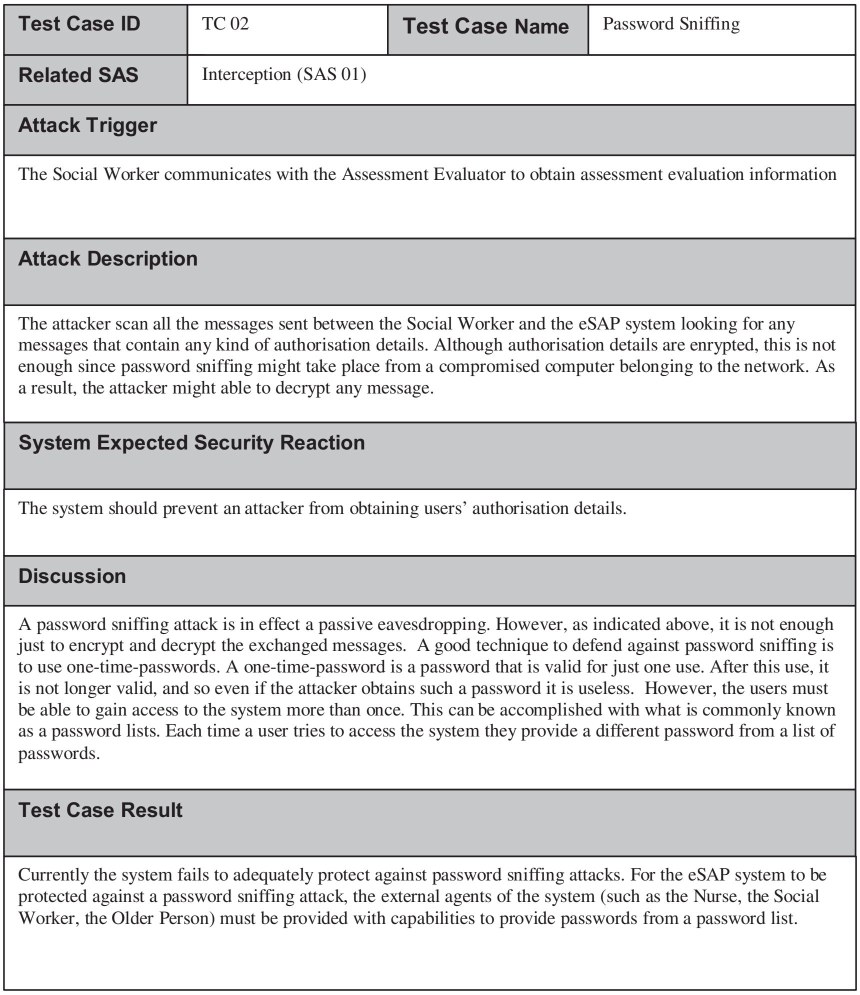
4.3.3. Apply the security test case

The test cases are applied and a decision is formed as to whether the system can prevent the identified attacks or not. The decision whether an attack can be prevented (and in what degree) or not lies on the developer. However as an indication of the decision it must be taken into consideration that at least one secure capability must help an attack, in order for the developer to decide the attack can be prevented. Attacks that cannot be prevented are notated on the



Test Case ID	TC 01	**Test Case Name**	Eavesdropping
Related SAS	Interception (SAS 01)		
Attack Trigger			
The Social Worker communicates with the Assessment Evaluator to obtain assessment evaluation information			
Attack Description			
The Attacker eavesdrops the information exchanged between the social worker and the assessment evaluator.			
System Expected Security Reaction			
The system should prevent Attacker from reading the information exchanged between the external and the internal actors.			
Discussion			
The Attacker will try to eavesdrop data from any communications between the external agents and the eSAP system. However, currently the system and its external agents have capabilities to encrypt and decrypt data. As a result all the important data is transmitted across the network encrypted and therefore the attacker cannot read it just by eavesdropping. However, the Attacker might try to obtain (or sometimes even guess) the encryption key.			
Test Case Result			
The system’s internal and external actors have capabilities to protect against eavesdropping attacks.			

Fig. 10. Eavesdropping test case.



Test Case ID	TC 02	**Test Case Name**	Password Sniffing
Related SAS	Interception (SAS 01)		

Attack Trigger

The Social Worker communicates with the Assessment Evaluator to obtain assessment evaluation information

Attack Description

The attacker scan all the messages sent between the Social Worker and the eSAP system looking for any messages that contain any kind of authorisation details. Although authorisation details are enrypted, this is not enough since password sniffing might take place from a compromised computer belonging to the network. As a result, the attacker might able to decrypt any message.

System Expected Security Reaction

The system should prevent an attacker from obtaining users' authorisation details.

Discussion

A password sniffing attack is in effect a passive eavesdropping. However, as indicated above, it is not enough just to encrypt and decrypt the exchanged messages. A good technique to defend against password sniffing is to use one-time-passwords. A one-time-password is a password that is valid for just one use. After this use, it is not longer valid, and so even if the attacker obtains such a password it is useless. However, the users must be able to gain access to the system more than once. This can be accomplished with what is commonly known as a password lists. Each time a user tries to access the system they provide a different password from a list of passwords.

Test Case Result

Currently the system fails to adequately protect against password sniffing attacks. For the eSAP system to be protected against a password sniffing attack, the external agents of the system (such as the Nurse, the Social Worker, the Older Person) must be provided with capabilities to provide passwords from a password list.

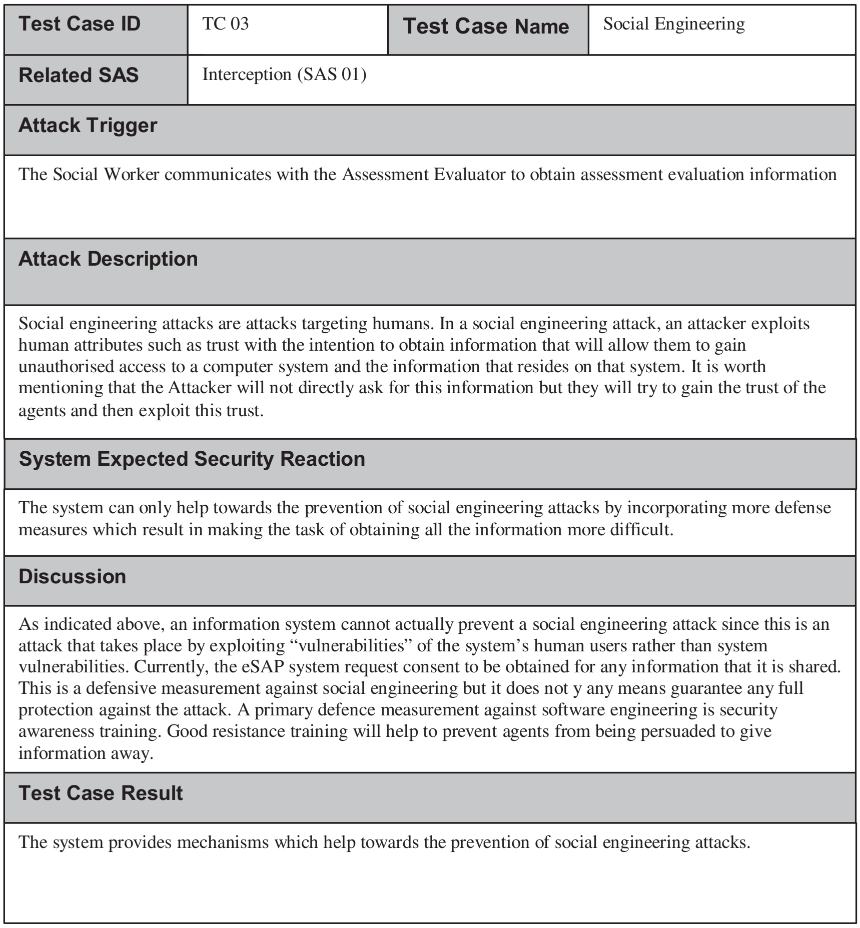
Fig. 11. Password Sniffing Test Case.

security attack scenarios as solid attack links, as opposed to attacks that the system can prevent and which are notated as dashed attack links. Consider for instance the eavesdropping attack introduced above. The Attacker will try to eavesdrop data from the communication between the external and the internal agents of the eSAP. However, the agents (both internal and external) have capabilities to encrypt and decrypt data. Therefore, a simple eavesdropping will not result in the attacker reading any of the transmitted data.

4.4. *Redefinition of system*

For each attack that it has been decided it cannot be prevented, as a result of failing test cases, extra capabilities must be assigned to the system to help towards the prevention of that attack. In general, the assignment of extra secure capabilities is not a unique process and it depends on the perception of the developer regarding the attack dangers. However, a good approach is to analyse the capabilities, related to the specific attack, of the attacker and assign capabilities to the actors' of the system that can revoke the attacker's capabilities.

For instance, the application of the above example test cases produced many useful results about the security of the eSAP system. First of all, it was identified that the system provides enough protection against some of the identified attacks. Secondly, for the attacks that the system did not provide adequate protection, extra agents and extra



Test Case ID	TC 03	**Test Case Name**	Social Engineering
Related SAS	Interception (SAS 01)		
Attack Trigger			
The Social Worker communicates with the Assessment Evaluator to obtain assessment evaluation information			
Attack Description			
Social engineering attacks are attacks targeting humans. In a social engineering attack, an attacker exploits human attributes such as trust with the intention to obtain information that will allow them to gain unauthorised access to a computer system and the information that resides on that system. It is worth mentioning that the Attacker will not directly ask for this information but they will try to gain the trust of the agents and then exploit this trust.			
System Expected Security Reaction			
The system can only help towards the prevention of social engineering attacks by incorporating more defense measures which result in making the task of obtaining all the information more difficult.			
Discussion			
As indicated above, an information system cannot actually prevent a social engineering attack since this is an attack that takes place by exploiting "vulnerabilities" of the system's human users rather than system vulnerabilities. Currently, the eSAP system request consent to be obtained for any information that it is shared. This is a defensive measurement against social engineering but it does not y any means guarantee any full protection against the attack. A primary defence measurement against software engineering is security awareness training. Good resistance training will help to prevent agents from being persuaded to give information away.			
Test Case Result			
The system provides mechanisms which help towards the prevention of social engineering attacks.			

Fig. 12. Software Engineering Test Case.

secure capabilities were identified and the following modifications took place in the eSAP system.

1. Capabilities were given to the external agents and to the Cryptography Manager to enable them to change the cryptographic algorithm often. The lack of such capabilities was identified during the read data test case of the interception attack scenario.
2. The external agents of the system were given the capability to provide passwords from a password list, and the Authenticator was given capabilities to successfully process such passwords. The lack of such capabilities was identified by the application of the password-sniffing test case of the interception attack scenario.

As explained earlier (see Fig. 3), when extra components need to be added to the system architecture the SAT process needs to be re-applied.[3]

5. Related work

The literature provides many references to scenario-based approaches for analysing and designing IS. However, just few proposals introduce the use of scenarios for testing and the derivation of test cases [23]. Hsia et al. [35] describe a method to create and validate scenarios which are used for acceptance testing. Michailova et al. [4] developed a scenario-based testing method for object-oriented programs that uses constraints. Similarly, Tsai et al. [36] proposed the SOOFT method, a scenario-based object-oriented test framework for adaptive and rapid testing. The SCENTOR approach [37]

[3] To keep the size of the paper to a minimum, we do not explain the second application of the SAT. This is explained in [22].

provides e-business-specific support for the generation of scenario-based tests using JUnit as a basis. All these approaches have been found valuable. However, they demonstrate some limitations. For instance, the Hsia approach does not provide activities for the derivation of concrete test cases and it requires extensive knowledge of formal methods since it is based on regular grammars and conceptual state machines. Similarly, the Michailova, the SOOFT and the SCENTOR also lack a well defined and structure approach for the derivation of scenarios and associated test cases. Moreover, these approaches only support implementation based testing, since their concepts and methods do not support testing during the earlier stages of the development process.

On the other hand, the SCEnario-Based Validation and Test of Software (SCENT) method [23] provides a step-by-step approach to formalise scenarios through state charts and then uses these state charts to create test cases. However, SCENT's ontology and methods do not support the development of security-related scenarios and test cases. In fact, as the developers of the method point out: "…integration of non-functional requirements [such as security requirements] in scenarios and statecharts is problematic…".

The work most related to ours is the attack scenario analysis method presented by Liu et al. [11]. However our work differs in important issues:

- Liu's et al. approach does not provide a structure process for creating and validating the attack scenarios. Therefore, developers find little help when employing that method. In contrast, our approach provides a well-defined process containing activities which assist the developer in identifying, creating and validating the security attack scenarios.
- Our approach provides a process to derive concrete test cases from the security attack scenarios to test the security of the system. In their approach, Liu et al. only use attack scenarios to identify security requirements and do not provide any support for deriving concrete test cases to test the security of the system. This limits the applicability of their attack scenario analysis method.
- Our security attack scenarios incorporate analysis of security countermeasures against the potential attacks. Guidance is provided, through defined activities, to assist developers in identifying countermeasures of the system and identify how these might influence the goals of the attacker. In their work, Liu et al., argue that when the intentions of the attackers are identified the system can be equipped with countermeasures. However they do not explain how such countermeasures can be identified neither they describe any process for applying these countermeasures to the system.

6. Conclusions

In this paper we have presented a novel scenario-based process that enables information system developers to test the security of the system under development during design time. We have also demonstrated the applicability of our approach by applying it to a real-life case study.

It is important to note that our process does not aim to replace existing security testing techniques which focus on testing the security of an information system after the implementation of the system. At design time we are not able to detect or consider attacks that are related to specific implementations. On the contrary, our process aims to complement such implementation-related testing techniques and provide a well-guided approach that enables developers to (1) identify important security vulnerabilities related to the design models of the system at an early stage in the developmental process; and (2) provide mechanisms to refine the system in order to overcome identified security vulnerabilities and to ensure that the design of the system enforces the necessary security requirements.

Nevertheless, further work is needed. The proposed approach has been applied to a case study from the health and social care sector. Therefore, an obvious direction for future work is the application of the approach to case studies from different sectors in order to obtain a better understanding of how the approach can be applied to different types of problems with different types of security attacks and security challenges. Moreover, the development of a tool to automate some of the processes of the approach would be an interesting and very useful direction for future work. Such a tool, not only will speed up the testing process but it will also allow developers not familiar with some aspects of the approach to successfully apply it by automating some of the process's activities.

References

[1] V.P. Lane, Security of Computer Based Information Systems, Macmillan Education ltd, 1985.

[2] R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, Wiley Computer Publishing, 2001.

[3] P. Devanbu, S. Stubblebine, Software engineering for security: a roadmap, in: Proceedings of the Conference of the Future of Software Engineering, 2000.

[4] A. Michailova, M. Doche, M. Butler, Constraints for scenario-based testing of object-oriented programs, Technical Report, Electronics and Computer Science Department, University of Southampton, 2002.

[5] A. Van Lamsweerde, Elaborating security requirements by construction of intentional anti-models, in: Proceedings of the International Conference on Software Engineering, 2004, pp. 148–157.

[6] J. Viega, G. McGraw, Building Secure Software—How to Avoid Security Problems the Right Way, Addison-Wesley, Reading, MA, 2004.

[7] J. McDermott, C. Fox, Using abuse care models for security requirements analysis, in: Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.

[8] M. Schumacher, U. Roedig, Security engineering with patterns, in: the Proceedings of the 8th Conference on Pattern Languages for Programs (PLoP 2001), Illinois, USA, September 2001.

[9] A.I. Anton, J.B. Earp, A requirements taxonomy for reducing web site privacy vulnerabilities, Requirements Eng. 9 (3) (2004) 169–185.

[10] J. Jurjens, Secure Systems Development with UML, Springer, 2004.

[11] L. Liu, E. Yu, J. Mylopoulos, Analysing security requirements as relationships among strategic actors, in: Proceedings of the 2nd Symposium on Requirements Engineering for Information Security (SREIS'02), Raleigh–North Carolina, 2002.

[12] H. Mouratidis, P. Giorgini, G. Manson, An ontology for modelling security: The tropos approach, knowledge-based intelligent information and engineering systems, Lecture Notes in Artificial Intelligence, vol. 2773, 2003.

[13] R. Crook, D. Ince, L. Lin, B. Nuseibeh, Security requirements engineering: when anti-requirements hit the fan, in: Paoceedings of the 10th International Requirements Engineering Conference, IEEE Press, 2002, pp. 203–205.

[14] H. Mouratidis, P. Giorgini, G. Manson, Integrating security and systems engineering: towards the modelling of secure information systems, in: Proceedings of the 15th International Conference on Advanced Information Systems (CaiSE), 2003.

[15] H. Mouratidis, P. Giorgini, G. Manson, When Security Meets Software Engineering: A Case of Modelling Secure Information Systems, Inf. Syst. 30 (8) (2005) 609–629.

[16] H. Mouratidis, G. Weiss, P. Giorgini, Modelling secure systems using an agent oriented approach and security patterns, Int. J. Software Eng. Knowledge Eng. 16 (3) (2006) 471.

[17] M. Bishop, Introduction to Computer Security, Addison-Wesley, Reading, MA, 2005.

[18] M.R. Blackburn, R.D. Busser, A.M. Nauman, R. Chandramouli, Model-Based Approach to Security Test Automation, in: Proceedings of Quality Week, 2001

[19] H. Mouratidis, I. Philp, G. Manson, A novel agent-based system to support the single assessment process of older people, J. Health Inf. 9 (3) (2003) 149–162.

[20] P. Brescianni, P. Giorgini, H. Mouratidis, G. Manson, Multiagent Systems and Security Requirements Analysis, in Advances in Software Engineering for Multiagent Systems, in: C. Lucena, A. Garcia, A. Romanovsky, J. Castro, P. Alencar (Eds.), Lecture Notes in Artificial Intelligence, vol. 2940, Springer, Berlin, 2003.

[21] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, TROPOS: an agent-oriented software development methodology, J. Autonom. Agents Multi-Agent Systems 8 (3) (2004) 203–236.

[22] H. Mouratidis, A security oriented approach in the development of multiagent Systems: applied to the management of the health and social care needs of older people in England, Ph.D. thesis, University of Sheffield, 2004.

[23] J. Ryser, M. Glinz, SCENT—a method employing scenarios to systematically derive test cases for system test, Technical Report 2000.03, Institut für Informatik, University of Zurich, 2000.

[24] C. Potts, K. Takahashi, A.I. Anton, Inquiry based requirements analysis, IEEE Software 11 (2) (1994) 21–32.

[25] A.I. Anton, W.M. McCracken, C. Potts, Goal Decomposition and Scenario Analysis in Business Process Reengineering, in: Proceedings of the 6th Conference on Advanced Information Systems (CAiSE-1994), Utrecht-The Netherlands, 1994.

[26] J.M. Carroll, M.B. Rosson, Getting around the task-artifact cycle: how to make claims and design by scenario, IBM Research Report, Human Computer Interaction, RC 17908 (75365), 1991.

[27] V. Lalioti, C. Theodoulidis, Visual scenarios for validation of requirements specification, in: Paoceedings of the 7th International Conference on Software Engineering and Knowledge Engineering (SEKE'95), Rochville, Maryland-USA, 1995.

[28] W. Stallings, Cryptography and Network Security: Principles and Practice, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.

[29] B. Schneier, Secrets & Lies: Digital Security in a Networked World, Wiley, New York, 2000.

[30] G. Kosters, B.U. Pagel, M. Winter, Coupling use cases and class models, in: Proceedings of the BCS-FACS/EROS workshop on "Making Object Oriented Methods More Rigorous," Imperial College, London-England, 1997.

[31] R.W. Selby, V.R. Basili, F.T. Baker, Cleanroom software development: an empirical Evaluation, IEEE Trans. Software Eng. 13 (9) (1997) 1027–1037.

[32] I. Sommerville, Software Engineering, seventh ed., Addison-Wesley, Reading, MA, 2005.

[33] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 729, 1990.

[34] R. Patton, Software Testing, Sams, 2000.

[35] P. Hsia, D. Kung, C. Sell, Software requirements and acceptance testing, Ann. Software Eng. 3 (1997) 291–317.

[36] W.T. Tsai, A. Saimi, L. Yu, R. Paul, Scenario-based object oriented testing framework, in: Proceedings of the 3rd International Conference on Quality Software, 2002, p. 410.

[37] J. Wittevrongel, F. Maurer, SCENTOR: scenario-based testing of e-business applications, in: Proceedings of the 10th IEEE International Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises, 2001, USA, p. 41.