J Ambient Intell Human Comput (2015) 6:361–373
DOI 10.1007/s12652-015-0269-8

ORIGINAL RESEARCH

Generating attacks in SysML activity diagrams by detecting attack surfaces

Samir Ouchani[1] · **Gabriele Lenzini**[1]

Received: 10 November 2014 / Accepted: 15 January 2015 / Published online: 18 March 2015

Abstract In the development process of a secure system is essential to detect as early as possible the system's vulnerable points, the so called *attack surfaces*, and to estimate how feasible it would be that known attacks breach through them. Even if attack surfaces can be sometimes detected automatically, mapping them against known attacks still is a step apart. Systems and attacks are not usually modelled in compatible formalisms. We develop a practical framework that automates the whole process. We formalize a system as SysML activity diagrams and in the same formalism we model libraries of patterns taken from standard catalogues of social engineering and technical attacks. An algorithm that we define, navigates the system's diagrams in search for its attack surfaces; then it evaluates the possibility and the probability that the detected weak points host attacks among those in the modelled library. We prove the correctness and the completeness of our approach and we show how it works on a use case scenario. It represents a very common situation in the domain of communication and data security for corporations.

Keywords Systems attacks · Attack patterns · Attack surfaces · SysML activity diagrams · Socio-technical security

✉ Samir Ouchani
samir_ouchani@yahoo.com; samir.ouchani@uni.lu

Gabriele Lenzini
gabriele.lenzini@uni.lu

[1] Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg, Luxembourg

1 Introduction

There are two distinct yet related challenges in the development of secure software and systems. One is about discovering vulnerabilities at as early as possible stages of the system development's life-cycle; the other is about assessing and possibly quantifying the degree of vulnerability of an existing system when this is exposed to known attacks.

A response to the first challenge requires to check whether a model of the system satisfies a set of relevant security properties. This check is performed in the presence of an attacker, usually a Dolev Yao adversary (Dolev and Yao 1983) that controls all the system's communication channels to interfere with the system's functionalities. This technique of analysis is known as *model checking* (Clarke et al. 1983). It has been successfully applied to discover insidious attacks and anomalies for the risk analysis and security assessment of the model-based systems (Solhaug and Seehusen 2014). However, although efficiently implemented in specific cases (Clarke et al. 2012), model checking's worst-case time complexity is exponential in the size of the system's and of the property's models. Large systems may be beyond reach for this type of analysis. The response to the second challenge, instead, is more pragmatic. Considering only documented attacks—that is, patterns of actions known to be used to compromise the system's integrity, availability, and confidentiality (Abrams 1998)—it consists in estimating the system's degree of vulnerability looking at the system's *attack surfaces* (Manadhata and Wing 2011). An attack surface is roughly the set of system's actions that are accessible externally and the system's resources which can be modified via those actions. The more extensive the attack surface is, the more vulnerable the system can be.

This paper's work is about this second challenge. It proposes a formal framework to detect attack surfaces *automatically* on systems modelled in SysML (OMG 2007a). SysML is a UML2.0-based formalism and a prominent object-oriented graphical language which has become *de facto* a standard in software and systems modelling. Assuming that a system is modelled in SysML is therefore a pragmatic choice, in order to be compliant with the current engineering practices. SysML reuses a subset of UML packages (OMG 2007b), namely the subset already extensively used in modelling large and complex systems, and it extends it with other features of quantitative nature, such as probability. These diagrams can call and communicate with other diagrams and allow for probabilistic behaviour specification. Particularly, SysML's *activity diagrams*, the specific formalism that this work adopts, can express a qualitative and quantitative elements of a system's behaviour and at various levels of abstraction (Holt and Perry 2008).

Detecting attack surfaces requires to inspect a system's model and to find out if known attacks can reach the system's core procedures via the system's exposed actions. The literature offers a variety of ways to describe attacks: attack tree (Mauw and Oostdijk 2005), attack graph (Sawilla and Ottawa 2007), and network attack graph (Sheyner 2004), to cite a few. Such models are used by many organizations that have a special interest in collecting, describing, and classifying attack patterns. For instance, large taxonomies of comprehensive samples of existing attacks have been built by security organizations, such as the common attack pattern enumeration and classification (CAPEC)[1], a sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security, and the web application security consortium (WASC)[2]. But existing techniques to model attacks are not compatible with the techniques commonly used to model systems. These last are not meant to be used to highlight easily a system's attack surfaces. Systems, on their side, are not modelled to be interfaced with attack trees/graphs/networks. There is so a gap between the way attacks and the way systems are modelled. Detecting attack surfaces against attack models is therefore not a process that can be done fully automatically.

This paper proposes a solution to this problem. We model both systems and attack patterns directly in SysML, and we design an algorithm that, by traversing a system model, collects all the system's attack surfaces and links them to the given library of attack patterns. We consider, as a proof of concept, standard attacks among those proposed by CAPEC, and we show how to model both technical attacks and social engineering attacks in SysML: the result is a rich library of socio-technical attack templates in such a formalism. The library can be of course extended.

This paper is based on a previous conference paper that the authors have published (Ouchani and Lenzini 2014). However, it extends considerably that work: it models social engineering attacks, not studied in Ouchani and Lenzini (2014), it improves the algorithm for searching attack surfaces, and proves the algorithm's correctness and completeness with respect to the library of attacks in input. The whole presentation has also been restyled and improved. Finally, the current paper applies the framework to a new use case, more complex than the one presented in Ouchani and Lenzini (2014) and more interesting from a socio-technical perspective. It is, in fact, a typical scenario in the domain of data and communication security for ICT-based corporations.

Outline The remainder of this paper is organized as follows. Section 2 reviews the existing related work. Section 3 describes and formalizes SysML activity diagrams; and Sect. 4 presents the concept of attack patterns. The attack generation framework is detailed in Sect. 5. The experimental results are described in Sect. 6. Finally, Sect. 7 concludes this paper and provides the future works.

2 Related work

In this section, we survey the existing initiatives related to system attacks modelling and to attack surfaces detection.

2.1 Attack modelling

A risk-based approach has been proposed to create modular attack trees for each component in the system (Grunske and Joyce 2008). These trees are specified as parametric constraints, which allow quantifying the probability of security breaches that occur due to internal and external component vulnerabilities. Another approach models probability metrics based on attack graphs as a special Bayesian network (Frigault and Wang 2008). Each node of the network represents vulnerabilities as well as the pre and post conditions. Jürjens and Shabalin (2004) and Houmb et al. (2010) extract specific cryptography-related information from UMLsec diagrams. Moreover, the Dolev–Yao model of an attacker is included with UMLsec to model the interaction with the environment. Further, Siveroni et al. (2010) extend UMLsec to model peer-to-peer applications along with their security aspects. They rely on the concept of abuse cases defined as UML use cases and state machine diagrams to represent attack scenarios. Morais et al. (2013) generate attack scenarios from the threat model of the wireless security protocol. First, they collect attacks from

[1] http://capec.mitre.org.

[2] http://www.webappsec.org.

vulnerabilities databases. Then, they classify them in terms of violated properties. Finally, they generate the protocol attack tree by relying to SecurelTree tool.

2.2 Attack surfaces detection

Gegick and Williams (2007) identify security vulnerabilities in code level by tailoring attack patterns based on the software components. These patterns take the form of regular expressions that are generic representations of vulnerabilities. Huang et al. (2011) distil attack surfaces of an attack graph by shifting out the minimum cost in the graph. They use SAT solver to view the minimum effort of an attack to conquer critical assets in the system. Vijayakumar et al. (2012) develop an approach based on runtime analysis to compute attack surfaces by finding the system adversaries in order to determine which program entry points access is an adversary controlled objects. They use the system's access control policy to distinguish adversary controlled data from trusted ones. Kantola et al. (2012) identify the communication attack surfaces by considering intent-based attacks on applications that do not hold common signature-level permissions. Any component of the correct type with a matching intent filter can intercept the intent. The possible attacks enabled by such unauthorized intent receipt depend on the type of the intent. Checkoway et al. (2011) analyse the external attack surface of modern automobile systems. Systematically, they synthesize the set of possible external attack vectors as a function of the attackers ability to deliver malicious input via specific modalities. For each modality, they characterize the attack surface exposed in current automobiles with their set of channels.

3 SysML activity diagrams

SysML (OMG 2007a) is a general-purpose, graphical, modelling language for specifying, designing, and verifying complex hardware and software systems, as well as organizative and procedural workflows. The language provides a semantic foundation for modelling a system structure and behaviour.

SysML *activity diagrams* are SysML's elements that focus on a system's behaviour. Activity diagrams are graphs: their vertices stand for activities (called *activity nodes*) and their edges stand for connections among activities (called *activity edges*) that define objects/data flow or control flows. In particular, an activity node can be of the following types:

- *An activity invocation element*: it sends or receives signals or objects, or it calls an operation or calls a behaviour.
- *A control flow element*: it defines the initial and the final flow of the diagram, or the final flow of a path, or a decision nodes. It can be a fork, a merge or a join node.

An activity edge can be of the following types:

- *A control flow element*: it shows the execution path through the activity diagram. Incoming edges are called *input edges*; outcoming edges are called *output edges*.
- *An object flow element*: it shows the object flow between activity nodes. Incoming edges are called *input tokens*; outcoming edges are called *output tokens*.

Branching is modelled with *decision nodes* and *merge nodes*. A decision node specifies a choice between different possible paths. The direction to take depends on the evaluation of a boolean guard, if the decision is boolean. It depends instead on a probability distribution, if the decision is probabilistic. A merge node specifies a point from where different incoming control paths start following the same path.

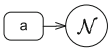
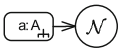
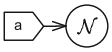
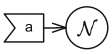
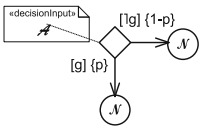
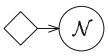
Concurrency and synchronization are modelled with *fork nodes* and *join nodes*. A fork node indicates the beginning of multiple parallel control threads. In UML2.0, on which SysML is based, fork nodes model unrestricted parallelism: thus, a token evolves asynchronously according to an interleaving semantics. A join node allows multiple parallel control threads to synchronize and rejoin.

Table 1 resumes the graphical artifacts of SysML activity diagrams (left column) and the corresponding formal expressions used to express the proposed framework (middle column) all followed by an informal description of the artifact (right column).

When a SysML activity diagram is invoked, its initial node activates. It is custom to assume that the initial node activates by possessing a token. A node activates, and thus it takes the token, only if the preceding node de-activates and if the condition guarding the node's incoming edge is satisfied. During execution, the action or the decision node that has an associated call behaviour can consume its input token and invoke its specified behaviour. SysML supports two types of invocations: synchronous and asynchronous. In the asynchronous invocation, the execution of the invoked behaviour proceeds without any further dependency on the execution of the activity that invokes it. In the synchronous invocation, the execution of the calling artifact is blocked until it receives a reply token from the invoked behaviour. In a decision node that has more than one path enabled, the choice of which behaviour to activate is done non-deterministically.

Definition 1 gives the formal definition of SysML activity diagrams. Properties 1 and 2 express how the structure and the control flow are constrained in a SysML activity diagram.

Table 1 Formalization of SysML Activity Diagram Artifacts

Artifacts	Formalization	Description
	$\iota \rightarrowtail \mathcal{N}$	Initial node. It is activated when a diagram is invoked
	$\odot$	Activity final node. It stops the diagram' execution
	$\otimes$	Flow final node kills its related path' execution.
	$a \rightarrowtail \mathcal{N}$	Action node defines an atomic action
	$a \uparrow \mathcal{A} \rightarrowtail \mathcal{N}$	Call behaviour node invokes a new behaviour
	$a!v \rightarrowtail \mathcal{N}$	Send node is used to send a signal/object
	$a?v \rightarrowtail \mathcal{N}$	Receive node is used to receive a signal/object
	$D(\mathcal{A}, p, g, \mathcal{N}, \mathcal{N})$	Decision node with a call behaviour $\mathcal{A}$ a convex distribution $\{p, 1-p\}$ and guarded edges $\{g, \neg g\}$
	$M(x) \rightarrowtail \mathcal{N}$	Merge node specifies the continuation where $x = \{x_1, x_2\}$ is a set of input pins
	$F(\mathcal{N}_1, \mathcal{N}_2)$	Fork node models the concurrency that begins multiple parallel control threads. UML 2.0 activity forks model unrestricted parallelism
	$J(x) \rightarrowtail \mathcal{N}$	Join node presents the synchronization where $x = \{x_1, x_2\}$ is a set of input pins

Definition 1 (*SysML Activity Diagram*) A SysML activity diagram is a tuple $\mathcal{A} = (\cdot, fin, \mathcal{N}, \mathcal{E}, \mathcal{K}, Prob, Tok)$, where:

1. $\cdot$ is the initial node,
2. $fin = \{\odot, \otimes\}$ is the set of final nodes,
3. $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$ is a finite set of activity nodes, where $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ are activity invocation, object and control nodes, respectively.
4. $\mathcal{E}$ is a finite set of activity edges,
5. $\mathcal{K}$ is a finite set of tokens,
6. $Prob : (\{\cdot\} \cup \mathcal{N}) \times \mathcal{E} \rightarrow \text{Dist}(\mathcal{N} \cup fin)$ is a probabilistic transition function that assigns for each node a discrete probability distribution $\mu \in Dist(\mathcal{N} \cup fin)$,
7. $Tok : \mathcal{N} \cup \mathcal{E} \rightarrow \mathcal{K}$ is a function that assigns for each node or edge one token.

Property 1 (*Structure Constraint*) For a SysML activity diagram $\mathcal{A}$, let $|\mathcal{E}|$ be the number of edges, and $|\mathcal{N}| = |\mathcal{N}_1| + |\mathcal{N}_2| + |\mathcal{N}_3|$ is the number of nodes. We have:

1. If $\mathcal{N}_3 = \emptyset, then : |\mathcal{N}| = |\mathcal{E}| - 1$
2. If $\mathcal{N}_3 \neq \emptyset, then : |\mathcal{N}| < |\mathcal{E}| - 1$

Property 2 (*Token Constraint*) In a SysML activity diagram $\mathcal{A}$, let $|\mathcal{E}|$ represents the number of edges, and $|\mathcal{K}|$ is the number of tokens. Then: $|\mathcal{K}| < |\mathcal{E}| + |\mathcal{N}|$.

4 Modelling attack patterns

The standard schema for describing attack patterns we refer to is that devised by the software assurance strategic initiative CAPEC, the already mentioned common attack patterns enumeration and classification. The schema consists of *primary* and *supporting* elements. The primary schema elements provides the following information: an attack pattern ID, the description of the attack, its related weaknesses, its typical severity, likelihood of exploitation, and the attack's abstraction level. The supporting schema elements gives the description, the diagnosis, and other enhancing information about the attack.

Inspired by these schemata we propose a SysML activity diagram attack pattern template (see Fig. 1). Each concrete attack pattern will be built by instantiating this template. The instantiation specifies the call behaviour action that is denoted by "Pattern Behaviour" in Fig. 1. The template's main control flow is a probabilistic decision. The probability that the attack does occur is P, whereas 1-P is the probability that the attack does not occur.

The value P is estimated on the basis of the "typical likelihood of exploitation" schema element provided within CAPEC catalogue. However, this schema element is a qualitative description of the likelihood (called CAPEC *term*), that ranges from "low" to "high" (see Table 2). In order to quantify these attributes, we propose to assign

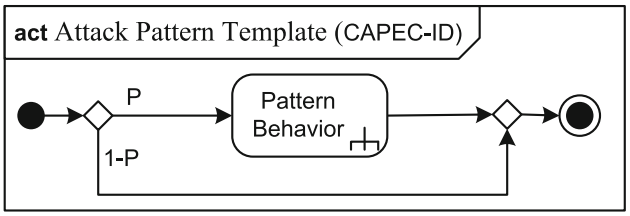


Fig. 1 The SysML activity diagram of the attack pattern template

ranges of probabilities to each qualitative description based on the *standard of security risk management* (ISO 2008) in combination with the *Kent's Words of Estimative Probability* (Sherman and the Board of National Estimates 2008), which proposes seven grades of likelihood. We combine the two schemes, and we propose the probabilities ranges as in Table 2.

The probability related to the instantiated pattern is obtained by the average of the probability interval assigned to a CAPEC term.

4.1 Instantiating the attack pattern: technical attacks

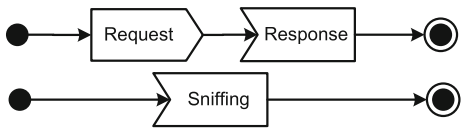
There are two categories of attacks that we considered relevant: *software attacks* (CAPEC-513), and *communications attacks* (CAPEC-512). The former is composed of twenty five attacks, among which the brute force (CAPEC-112), authentication abuse (CAPEC-114). We do not list the all attacks, for sake of space, but they can be found in the CAPEC taxonomy. The latter includes two attacks: interception (CAPEC-117) and protocol manipulation (CAPEC-272). In the following, we model a selected set of these technical attacks.

- Spoofing (CAPEC-156): an attacker builds a message to masquerade an authorized message from a trusted principal. Consumers of these messages can be manipulated into responding or processing the deceptive message. This attack may refer to spoofing the message's content (CAPEC-148) or to spoofing the message's senders or receivers (CAPEC-151). Their pattern is depicted by the following figure such that P (CAPEC-148) = P (CAPEC-151) = 0.8. We have assigned the value of 0.8 since their severity is high. This means that the average of 60 and 100 %.

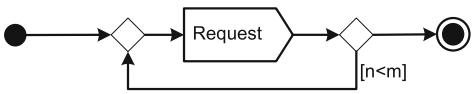
- Data leakage (CAPEC-118): the attacker uses well-formed requests to get sensitive information by exploiting weaknesses in the design. Three techniques are used in this class: data excavation (CAPEC-116), data interception (CAPEC-117), and sniffing (CAPEC-148). CAPEC-116 and CAPEC-117 are presented by the first control flow with P (CAPEC-116) = 0.5 and =P (CAPEC-117) = 0.5. Also, CAPEC-148 is illustrated in the second control flow with a probability value P (CAPEC-148) = 0.2.

Table 2 Probability values scale

CAPEC terms	Kent's estimative terms	Probability values
High	Certain	100
	Almost certain	93 % (±6 %)
Medium to high	Probable	75 % (±12 %)
Medium	Chances about even	50 % (±10 %)
Low to medium	Probably not	30 % (±10 %)
Low	Almost certainly not	7 % (±5 %)
	Impossible	0

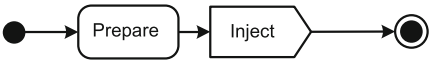


- Resource depletion (CAPEC-119): the attacker depletes a resource to the point that the target's functionality is affected. The result is usually the degradation or denial of one or more services offered by the target. The attacker can achieve his objective through flooding (CAPEC-125), through leak (CAPEC-131) by uploading a malicious file, or through allocation (CAPEC-131) by sending a formatted request. The pattern of these attacks is depicted by the following diagram where n is the number of requests and m is a number fixed by the designer. These attacks are launched by these probability values: P(CAPEC-125) = 0.8, P(CAPEC-131) = 0.8.

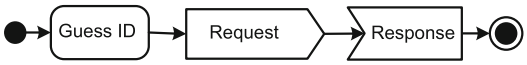


- Injection (CAPEC-152): The attacker is able to control or disrupt the behaviour of a target through crafted input data submitted using an interface functioning to process data input. Different resource-dependent patterns are detailed in CAPEC and abstracted to design level such as SQL (CAPEC-66), email (CAPEC-134), format string (CAPEC-135), LDAP (CAPEC-136), resource injection (CAPEC-240), script injection (CAPEC-242), and command injection (CAPEC-248).

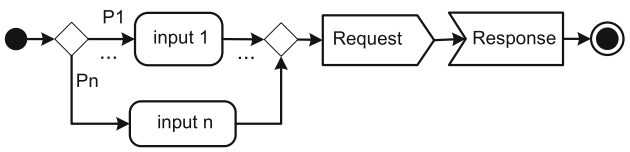
All of them take the form of the following control flow with a probability of 0.8.



- Exploitation of authentication (CAPEC-225): the attacker exploits the weaknesses related to authentication mechanisms including authentication bypass by spoofing (CWE-290), authentication bypass by assumed immutable data (CWE-302), and origin validation error (CWE-346). Particularly, its descendant sub-category CAPEC-21 aims at exploiting session variables, resource IDs and other trusted credentials to exploit that some software accepts user input without verifying its authenticity. They have the following work flow with P = 0.8.



- Fuzzing (CAPEC-28): of the probabilistic techniques (CAPEC-223) and it is inspired by a software testing method. The attacker provides randomly generated input to the system and looks for an indication to identify weaknesses in the system. The pattern of this attack is depicted by the following control flow such that P(CAPEC-28) = 0.8 and P_1, $P_2, \ldots$, P_n are probability values (e.g. uniformly distributed).



4.2 Instantiating the attack pattern: social engineering attacks

In this category fall attacks that use social engineering techniques. Social engineering attacks target people and persuade them to perform actions, usually divulging confidential information that should not be shared. In this way social engineering can gain access to a computer system and its resources without hacking the system.

The CAPEC taxonomy has recently included social engineering attacks too. From it we propose the classification shown in Fig. 2. It consists of three categories of attacks: *social information gathering* (CAPEC-404), *information elicitation* (CAPEC-410), and *target influence via social engineering* (CAPEC-416). Each category has a set of attacks and sub-class of attacks.

We select one of such attacks and we show how to model it in SysML: spear phishing (CAPEC-163), which uses phishing (CAPEC-98).

Spear phishing is a very common social engineering attack. In fact, recent statistics show that 91 % of attacks are phishing Corporation (2014), and about 95 % of cyber espionage attacks started with a phishing email. By modelling it therefore we have maximum relevance and deceptive capability.

In spear phishing, the intruder starts by obtaining useful information about the targeted user or organization. First, the intruder conducts a web searching and identifies trusted associates of target, pretexts the users, and collects social information via dumpster diving, and traditional and non-traditional sources. Then, the intruder creates a domain name that looks similar to the legitimate one and a legitimate SSL certificate for the new domain name. After, the attacker develops a duplicate of the legitimate website, for example, by use spidering software. The website may include very specific user information such as local temperature. Then, the intruder sends to the user a message from a spoofed legitimate-looking e-mail address or post a phishing link in an online forum that asks the user to click on the included link. After that, the intruder convinces the user to enter sensitive information on attacker's site. Finally, the intruder uses the stolen credentials to log into legitimate site.

Based on spear phishing description, the pattern of this attack is depicted by the diagrams depicted in Fig. 3. The first diagram has three call behaviour actions where each one calls the appropriate diagram with respect to the diagrams order. Since in CAPEC, the typical severity and the likelihood of this attack are evaluated high, then, the probability value to launch this attack is: P(CAPEC-163) = 0.8.

5 Attacks generation framework

In this section, we detail our proposed framework that automatically finds attacks that match a given system. The schema of the framework is depicted in Fig. 4. It takes as input a system modelled by a set of SysML activity diagrams that can be designed either by relying on the system specification document, or by reverse-engineering the system source code. To generate attacks specific to the system under test, the framework uses the library of attack templates that is proposed in Sect. 4. Then, the framework proposes an algorithm to detect attack surfaces from where an attack can damage the system. Further, the algorithm assigns for each detected attack surface a set of potential attacks. Based on them, the framework produces the possible application-dependent attacks that are instantiated from the attack library. As a result, a set of concrete attacks proper to the system under test is produced.

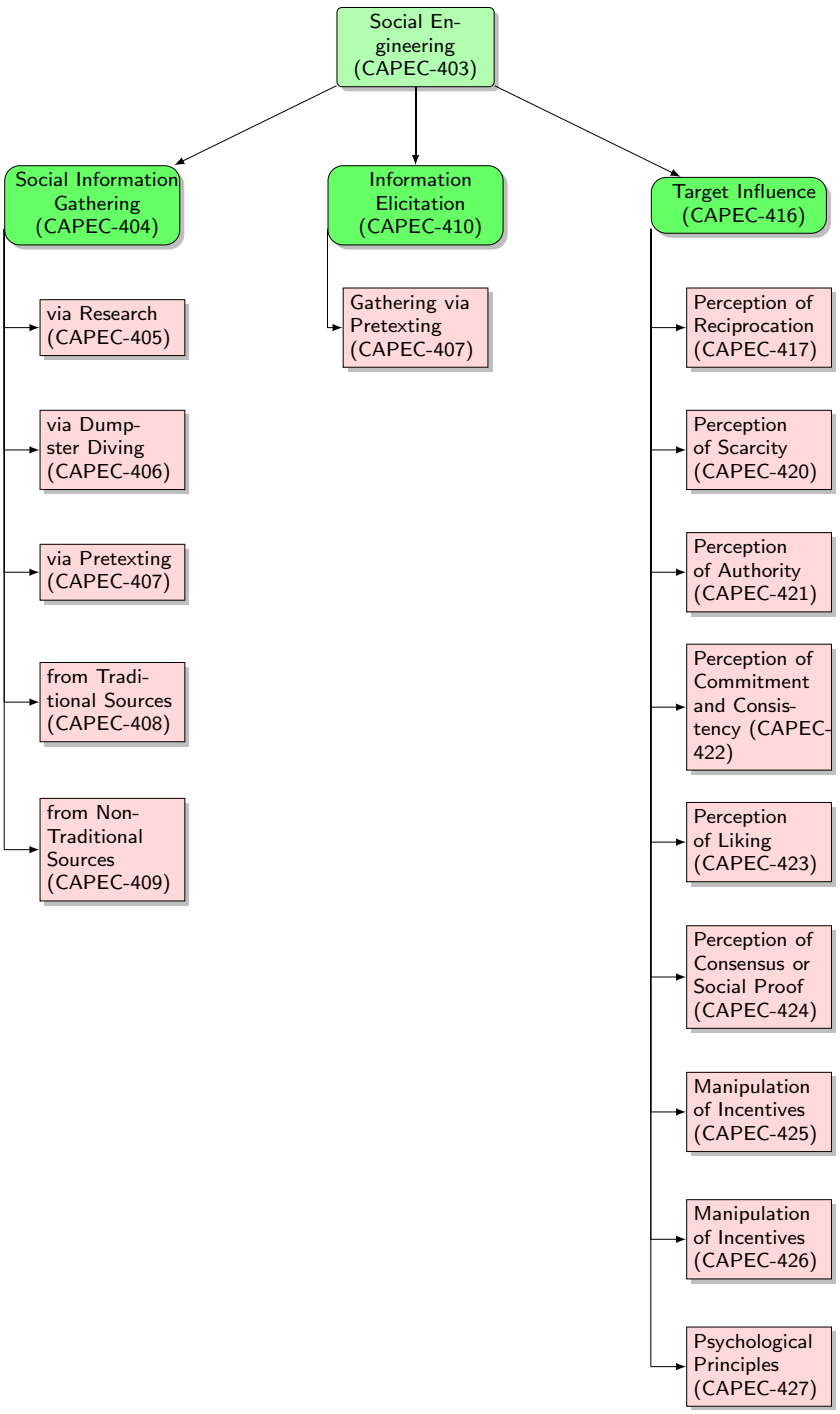


Fig. 2 CAPEC taxonomy of social engineering attacks

5.1 Attack surfaces detection

A system's attack surface relates with a system's exposed vulnerabilities, and then with an adversary's ability to interfere with the system and to damage it. The larger the attack surface, the greater the vulnerability, the more potential attacks the system may suffer. An attack surface is described as a subset of a system's resources, usually the system's data, variables, and actions, that an intruder can control and, in so doing, to interfere with the system's behaviour.

A key notion is that of *untrusted objects*.

Definition 2 (*Untrusted object*) An object v is *untrusted* if it is acquired by an input action or if it depends on an untrusted object. Object v *depends* on another object w, if v is calculated from w.

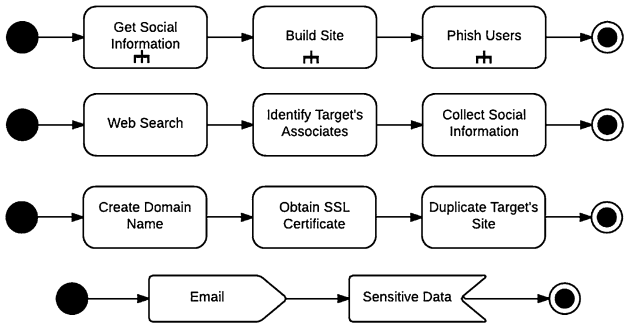


Fig. 3 Spear phishing template

Definition 3 formalizes the notion of attack surface in SysML terms.

Definition 3 (*Attack surface*) Let $\mathscr{A}$ be SysML model. An attack surface is a tuple $\omega = \langle N, X, O, Ch \rangle$, where:

1. N is the set of entry points of $\mathscr{A}$, all the artifacts except send artifacts.
2. X is the set of exit points of $\mathscr{A}$, the send artifacts.
3. O is the set of untrusted objects of $\mathscr{A}$.
4. $Ch : N \cup X \rightarrow 2^O$ maps entry or exit points to untrusted objects.

To determine the attack surfaces of a SysML activity diagram $\mathscr{A}$, we parse, depth-first manner, the diagram. Procedure Ξ, illustrated in Algorithm 1, does this search, and construct the attack surface $\Omega = (\omega_1, \omega_2, \omega_3, \omega_4)$. Here ω_1, ω_2, ω_3, and ω_4 are the entry points, the exit points, the untrusted data, and the channels, respectively.

Herein is the description of Algorithm 1. First, the initial node $\imath$ of $\mathscr{A}$ is pushed into the stack of nodes, denoted by *nodes* (line 11). While the stack *nodes* is not empty (line 12–39), the algorithm pops a node from the stack *nodes*, denoted by *cNode* (line 13). If *cNode* has a call behaviour (line 15), the called diagram is pushed in the stack *Beh* (line 16). Then, the current node is added into the list *vNode* of visited nodes (line 18), but only if it has not been visited already (line 14). Its successors are stored in the list *nNode* (line 19). The algorithm constructs the attack surfaces Ω (line 21, 24, 27, and 30) by checking the type of *cNode* (lines 20, 23, 26, and 30, respectively). Notably, objects that are inputted are inserted in the list of untrusted objects directly. Each artifact that contains an untrusted object or an objects that depends on any untrusted objects is added to the attack surface and linked to the untrusted object that determines its vulnerability. The artifact's related object(s), in turn, are added to the list of untrusted objects. The explored successors are pushed into the stack *nodes* (line 33–35), then, they are erased from the list *nNode* (line 36). The algorithm calls recursively itself (line 39) since there is a behavioural diagram in *Beh* (lines 38–40). Finally, the algorithm calls the function Λ (line 41) that assigns for each attack surface a set of attacks (detailed in the next section). Then, it terminates since all nodes of all diagrams are visited (lines 12 and 38).

Algorithm 1 Attack Surfaces Detection Algorithm.

Input: SysML activity diagrams $\mathscr{A}$.

Output: Attack Surfaces $\Omega = (\omega_1, \omega_2, \omega_3, \omega_4)$.

```
1:  Beh as Stack;                                   /* A stack of diagrams that is initially empty. */
2:  nodes as Stack;                                 /* A stack of nodes which is initially empty. */
3:  cNode as Node;                                  /* The current node which is initially empty. */
4:  nNode as list_of_Node;                          /* The list of new nodes that is initially empty. */
5:  vNode as list_of_Node;                          /* The list of the visited nodes that is initially empty. */
6:  ω1 as list_of_Node;                             /* A list of entry points that is initially empty. */
7:  ω2 as list_of_Node;                             /* The list of exit points that is initially empty. */
8:  ω3 as list_of_Object;                           /* The list of untrusted data that is initially empty. */
9:  ω4 as list_of_Tuple;                            /* The list of channels that is initially empty. */
10: procedure Ξ(A)
11:     nodes.push(A.ι);                            /* Push the initial node in the stack nodes. */
12:     while not nodes.empty() do
13:         cNode := nodes.pop();                   /* Pop the current node. */
14:         if cNode not in vNode then
15:             if cNode.type() in CallBeh then
16:                 Beh.push(cNode.Call());         /* Push the behaviour of cNode in the stack Beh. */
17:             end if
18:             vNode.add(cNode);                   /* Consider the current node as a visited node. */
19:             nNode := cNode.successors();        /* Get the successors of the current node. */
20:             if cNode.type() is an entry artifact then
21:                 ω1.add(cNode);                  /* Construct the set of the entry points. */
22:             end if
23:             if cNode.type() is an exit artifact then
24:                 ω2.add(cNode);                  /* Construct the set of the exit points. */
25:             end if
26:             if cNode.type() is an input artifact {a?v} then
27:                 ω3.add(cNode.object());         /* Add the object to the untrusted objects. */
28:             else
29:                 if cNode.object() is measured from at least a value in ω3 then
30:                     ω4.add((cNode,cNode.object())); /* Link the point to its untrusted object. */
31:                     ω3.add(cNode.object());     /* Add the object to the untrusted objects. */
32:                 end if
33:             end if
34:         end if
35:         for all n in nNode do                   /* Stores all newly discovered nodes. */
36:             nodes.push(n);
37:         end for
38:         nNode.clear();                          /* Empty the list nNode. */
39:     end while
40:     while not Beh.empty() do
41:         Ξ(Beh.pop());                           /* Call the function Ξ for a new behavioural diagram. */
42:     end while
43:     Λ(ω1 ∪ ω2 ∪ ω3 ∪ ω4);                       /* Call the function Λ */
44: end procedure
```

5.2 Application-dependent attacks generation

Our objective is to assign the appropriate attack template for each attack surface that is detected by Algorithm 1. Then, we instate this template to be dependent to the system under study. For that, we propose the function Λ that is described in Listing 1 which assigns for each attack surface $\omega \in \Omega$ at least one attack $k \in \mathbb{K}$. The set of attacks $\mathbb{K} = \{k_1, k_2, k_3, \ldots, k_{11}\}$ where the CAPEC id of a k_i is given by Table 3. For example, k_1 is CAPEC-148 and k_{11} is CAPEC-163.

Listing 1: Attack Surfaces Assigning Function.

```
1  Λ : Ω → 2^𝕂
2  Λ(ω) = ∀ω ∈ Ω, Case (ω) of
3              ι ↣ 𝒩            ⇒ in {k1,k6,k7,k11} ∪ Λ(𝒩) end
4              a ↣ 𝒩            ⇒ in {k2,k4} ∪ Λ(𝒩) end
5              a ↑ 𝒜 ↣ 𝒩        ⇒ in {k3,k4,k10,k11} ∪ Λ(𝒩) ∪ Λ(𝒜) end
6              a!v ↣ 𝒩          ⇒ in {k2,k3,k4} ∪ Λ(𝒩) end
7              a?v ↣ 𝒩          ⇒ in {k1,k8,k11} ∪ Λ(𝒩) end
8              D(𝒜,p,g,𝒩1,𝒩2)   ⇒ in {k5,k7,k9,k11} ∪ Λ(𝒩1) ∪ Λ(𝒩2) ∪ Λ(𝒜) end
9              M(x,y) ↣ 𝒩       ⇒ in {k5,k9} ∪ Λ(𝒩) end
10             F(𝒩1,𝒩2)         ⇒ in {k5,k9} ∪ Λ(𝒩1) ∪ Λ(𝒩2) end
11             J(x,y) ↣ 𝒩       ⇒ in {k5,k9} ∪ Λ(𝒩) end
12         otherwise(final activity or flow nodes) ⇒ in {} end
```

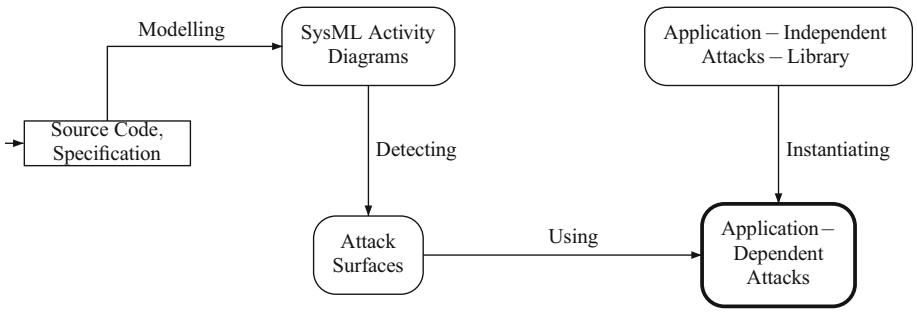


Fig. 4 System attacks generation framework

5.3 Correctness and complexity

Procedure Ξ search exhaustively all attack surfaces. Function Λ assigns to each detected attack surface the appropriate attacks from the library. Our algorithm is correct, in the sense that it points out all the model's attacks surfaces and assigns to each of them all the appropriate attacks among the ones in the attack library. Proposition 1 proves such statements.

Proposition 1 (Correctness and completeness)

(a) Algorithm Ξ is correct and complete, i.e. it detects all and only the $\mathscr{A}$'s artifacts that process or depend on untrusted objects.
(b) Algorithm Λ is correct and complete, i.e. it assigns to each node all and only the attacks that are applicable to the node.

Proof To prove (a) we argue that procedure Ξ is a depth-first search. It exhaustively parses all nodes in $\mathscr{A}$. The while loop (line 12–39) pushes in the stack '*nodes*' all $\mathscr{A}$'s nodes, pops them out one by one, and terminates only when the stack is empty. Test conditions in lines 20, 23, and 26 checks all $\mathscr{A}$'s artifacts, and build the attack surface. Statement of line 26 tests for objects that are untrusted because input objects. Line 29 checks artifacts for being related with untrusted objects, maps them together, and add those objects to the list of untrusted objects. Therefore Ξ correctly build set of entry points, the set of exit points, and the function that maps points to their untrusted objects.

To prove (b), we argue about Λ. The correctness of Λ is proved by induction on the structure of $\mathscr{A}$'s artifacts. The base case is obvious: Λ assigns to the final activity and flow nodes the empty set (see Listing 1), which is correct since no attack can be associated to those nodes. To prove that Λ assigns to the artifacts in the attack surface just build all and only the attacks that are applicable, we reason on case-by-case bases. But, here, we give the argument only for $a?v \rightarrow \mathscr{N}$: the other cases are similar and are omitted. Λ associates $a?v$ with attack k_1, k_8, and k_{11} (see Listing 1). All these attacks can send a message that will be received in $a?v$, thus they are applicable. Besides, no other attacks is applicable to this node. This can be proven by exclusion: we show only the case concerning k_7. The template of k_7 is a sniffing by a receive message, but no message can be received from the $a?v$ artifact. So k_7 is not applicable. To exclude all the other attacks we use similar arguments. By inductive hypothesis, $\Lambda(\mathscr{N})$ contains all and only applicable attacks and consequently $\Lambda(a?v \rightarrow \mathscr{N}) = \{k_1, k_8, k_{11}\} \cup \Lambda(\mathscr{N})$ contains all and only attacks which

Table 3 Simplification of CAPEC ids

k_i	1	2	3	4	5	6	7	8	9	10	11
CAPEC-id	148	151	116	117	125	131	148	152	225	28	163

are applicable to the artifact $a?v \rightarrow \mathscr{N}$. The argument about the correctness and completeness of Λ and the other artifacts is similar and we omit it.

Point (a) and (b) together prove the correctness and completeness of Algorithm 1.

Proposition 2 Algorithm 1 runs in time linear on the number of $\mathscr{A}$'s nodes.

Proof We count the number of steps of Ξ: the algorithm's time complexity is proportional to this number. Ξ traverses all $\mathscr{A}$'s nodes exactly once, and this takes $O(|\mathscr{N}|)$ steps, where $\mathscr{N}$ are $\mathscr{A}$'s nodes. Each tests/checks in Ξ can be done in constant time. The built attack surface is at most as large as $|\mathscr{N}|$, and so Λ, whose takes constant time to associate handful attacks to each nodes, runs $O(|\mathscr{N}|)$ time too.

6 Case study

We analyse a use case scenario in our framework. The scenario describes a very common situation. It regards a corporate organizations, such as a company or an university, whose employers use the information and communication technology (ICT) intensively to collaborate with outside, national or international, partners. They communicate prevalently per e-mails and share documents in repositories (or in the cloud) which they access remotely through a browser or by SVN clients.

depends on a secret password. There are many points where an attack can intrude. The password can be weak, and he can guess it, or it he can target the human user by sending phishing mails [e.g., (Francesco et al. 2013)] in the hope to victim replies by giving up the password directly or that he/she clicks on a link which will cause the download of a trojan or of a similar digital malwares that give control to the intruder. However, a pattern of an attack can be complex and not so easily discoverable as we will show. What we describe below has been taken from a real situation, one of the many that happens in the domain of communication and data security for ICT-based corporations. We have removed any reference to real names, ids and domain names but the core is exactly the same.

6.1 Description

A corporate organization works in a research project called "XSPARS". The project has a public web page, say http://www.xspars-project.eu, and a repository, available at https://www.xspars-project.eu/repositories/xspars. The repository access is protected by login and password. This is a common choice which, is weakened further, when is the host of the repository that generate the login and password for anyone, and distribute the pair by email.

During the execution of the project, the researchers that participate to the project receives emails, usually from a mailing list such as xspars-all@xspars-project.eu. A typical mail form that mailing list looks like the following:

```
Dear all,
Below are the link to the agenda of the next MT meeting.
Kindly let me know whether you would like to add anything to the agenda.

Agenda 24 October 2014
https://www.xspars-project.eu/repositories/XSPARS/trunk/Meetings/2014/Agenda.doc

Regards,
Alice
```

In such a scenario there are obvious concerns about the security of confidential data. What employers share in repositories may be sensitive information, as it happens when such corporations are collaborating in partnership projects. The access to such data should be protected by authentication protocol but often, despite not always, it all

Such mails, useful in their giving directly the link to access to the document, are very common. The text quoted above is actually taken from a real mail, posted in the mailing list of an existing project. The scenario also considers that xspars's researchers may read their mails using a tablet or a smart phone. With that device they also access

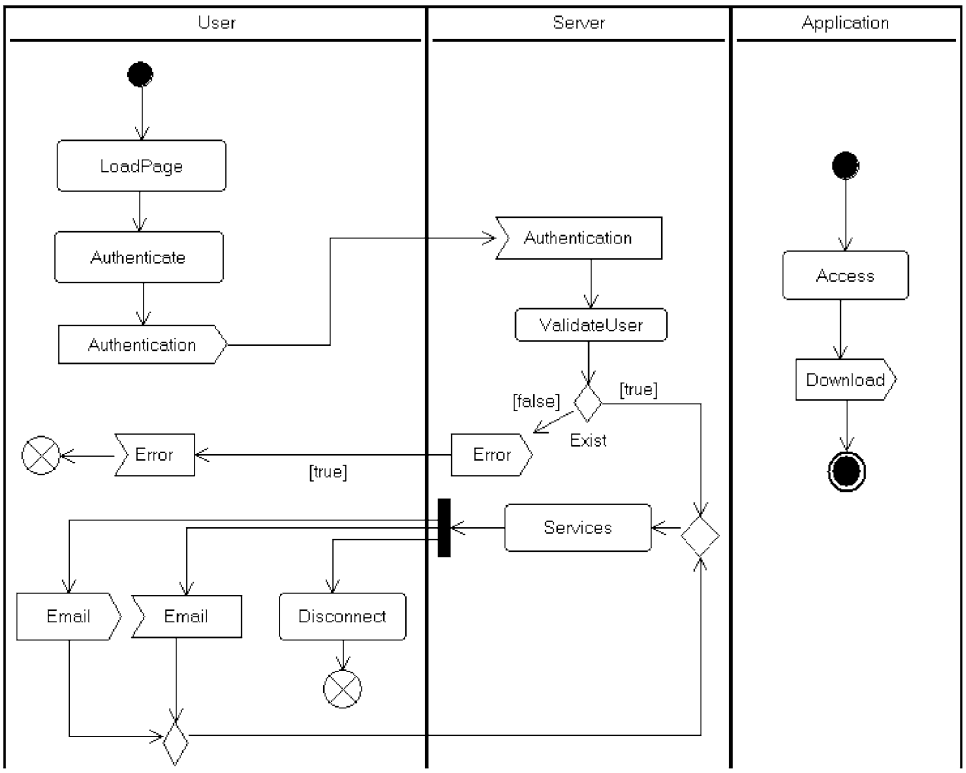


Fig. 5 XSPARS SysML activity diagram

the repository, download, read, write (or simply comment) and re-upload documents.

The system that is part of this scenario consists of users and their devices (a tablet or a smartphone), which run an email client and a browser. (Indeed only a browser is sufficient if we assume that the researchers are accessing their mail via a webmail server). The system also include two servers, the repository and the mail server. Figure 5 shows the SysML representation of a part of this system, that concerning the access to the repository. The part about reading emails is similar.

6.2 Analysis

Using our framework, the library of attacks, and the algorithm presented we have found the following attack surfaces. Each represents a possible attacks on the model of the system. One of such attack surface, (n, x, o, ch), is the following:

$$
\begin{aligned}
n &= \{\texttt{Email?emailbody}, \texttt{Loadpage}(\texttt{url})\} \\
x &= \{\texttt{Authentication!}(\texttt{url}, \texttt{login} - \texttt{password})\} \\
o &= \{\texttt{email} - \texttt{body}, \texttt{url}\} \\
ch &= \{(\texttt{Email?emailbody}, \texttt{emailbody}), (\texttt{Loadpage}(\texttt{url}), \texttt{url}), \\
&\quad (\texttt{Authentication!}(\texttt{url}, \texttt{login} - \texttt{password}), \texttt{url})\}.
\end{aligned}
$$

Here, we have give a name for the objects, but they are not represented in Fig. 5. Object `emailbody` is untrusted because that object is the object of an input artifact, and `url` is untrusted because it depends on it: the url is taken from the body of the email. Figure 6 shows an instance of the attack k_{11} (CAPEC-163), which is associated to n and to x.

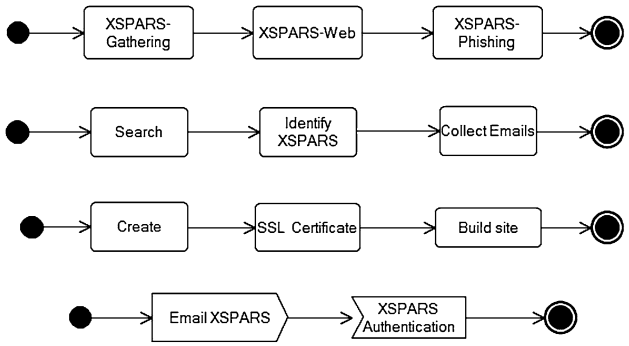


Fig. 6 SysML activity diagram of the attack for XSPARS system

An analysis of the attack surface, reveals that such an attack is actually possible. Here how it works. To prepare this attack, the intruder starts by doing a web search (Fig. 3, top). He finds the project's web page, and there the members of the project and their names. He collects the emails and information concerning the research topics. This second step of the attack is shown in the Fig. 3, middle. Knowing the public web page, the intruder clones it, creates a similar url, say https://www.xpsars-project.eu

and get a self-signed certificate for it. It also creates a repository there.

Then, it sends an email to one of the researcher with a text that mirrows the text of an usual mail with a link to the made-up repository https://www.xpsars-project.eu/repositories/XSPARS/Meetings/2014/Agenda.doc (Fig. 3, bottom).

The intruder counts to phish those researchers that may not be warned by browser's alerting them because of self-signed certificate [such as mini Opera, which only change the visual icon of the lock but does not warns explicitly the users (Bella et al. 2013)]. The intruder aims at having the researcher logging in into its repository. It can engineering the document in such a way to appear corrupt or empty, but from this action he can steal the researcher login-password. Here, reasoning further from what the attack surface shows, we imagine the intruder accessing the real repository and later mirroring the entire original repository into its own repository so to make it look like the real repository. In this way he can re-iterate the attack and hope to phish more researchers from the consortium.

Of course this analysis just made on the basis of the given attack surface gives an argument that the attack is feasible. It does not necessarily happen, or happen in this way. As a final remark, we note that, in this argument, we avoided to talk about probabilities. They may be not easily estimable.

7 Conclusion

One way to reduce the cost of system and software products is to detect vulnerabilities to attacks, technically called *attack surfaces*, at early stages of the development life-cycle. We presented a framework to detect attack surfaces and the attacks that can exploit the surfaces. We developed a library of system attacks that includes for the first time the social engineering aspects. In addition, we devised an algorithm that detects attack surfaces of the system and a function that assigns for each attach surface a set of potentially harmful attacks. We proved the correctness and the completeness of the whole procedure. We also validate the effectiveness and the efficiency of the presented framework by applying it on a real case which is a system of a research group from our institution. The results show the potentiality of presented approach.

The presented work can be extended in the following directions. First, we intend to apply our framework on different real cases. Also, we would like to achieve more complete catalogue that covers more type of attacks such that related to product chain and cyber-physical systems. In addition, as a next task is to implement the proposed algorithm and deliver a prototype.

Acknowledgments The research leading to the results presented in this work received funding from the Fonds National de la Recherche Luxembourg, project "Socio-Technical Analysis of Security and Trust", C11/IS/1183245, STAST, and the European Commissions Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318003 (TREsPASS).

References

Abrams MD (1998) Nims information security threat methodology. In: Mitre Technical Report MTR 98 W000009, MITRE, Center for Advanced Aviation System Development. McLean, Virgini

Bella G, Giustolisi R, Lenzini G (2013) A socio-technical understanding of TLS cerficate validation. In: Proceedings of 7th IFIP international conference on trust management (IFIPTM2013). Malaga. IFIP

Checkoway S, McCoy D, Kantor B, Anderson D, Shacham H, Savage S, Koscher K, Czeskis A, Roesner F, Kohno T (2011) Comprehensive experimental analyses of automotive attack surfaces. In: Proceedings of the 20th USENIX Conference on Security (SEC 11). USENIX Association, pp 6–6

Clarke EM, Emerson EA, Sistla AP (1983) Automated verification of finite state concurrent systems using temporal logic specifications: a practical approach. In: Proceedings of POPL, pp 117–126

Clarke EM, Klieber W, Novacek M, Zuliani P (2012) Model checking and the state explosion problem. In: Meyer B, Nordio M (eds) Tools for practical software verification. Lecture notes in computer science. Springer, Berlin

Dolev D, Yao AC (1983) On the security of public key protocols. IEEE Trans Inf Theory 29(2):198–208 (ISSN 0018–9448)

Francesco C, Ciaramella A, Staiano A (2013) Machine learning and soft computing for ict security: an overview of current trends. J Ambient Intell Humaniz Comput 4(2):235–247 (ISSN 1868–5137)

Frigault M, Wang L (2009) Measuring network security using Bayesian network-based attack graphs. In: Proceedings of the 32nd IEEE international computer software and applications conference (COMPSAC '08), pp 698–703

Gegick M, Williams L (2007) On the design of more secure software-intensive systems by use of attack patterns. Inf Softw Technol 49:381–397

Grunske L, Joyce D (2008) Quantitative risk-based "security prediction for component-based systems with explicitly modeled attack profiles. J Syst Softw 81:1327–1345

Holt J, Perry S (2008) SysML for systems engineering. Professional Applications of Computing Series 7, Institution of Engineering and Technology, London, UK

Houmb SH, Islam S, Knauss E, Jürjens J, Schneider K (2010) Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and UMLsec. Requir Eng 15:63–93 (ISSN 0947–3602)

Huang H, Zhang S, Ou X, Prakash A, Sakallah KA (2011) Distilling critical attack graph surface iteratively through minimum-cost sat solving. In: ACSAC'11, pp 31–40

Information technology, Security techniques, Information security risk management ISO (2008) International organization for standardization

Jürjens J, Shabalin P (2004) Automated verification of UMLsec models for security requirements. In: UML 2004. The unified modeling language, LNCS vol 2460. Springer, Berlin, pp 412–425

Kantola D, Chin E, He W, Wagner D (2012) Reducing attack surfaces for intra-application communication in android. In: Proceedings

of the 2nd ACM Work. On security and privacy in smartphones and mobile devices (SPSM 12), ACM, pp 69–80

Kent Sherman and Collected Essays the Board of National Estimates (2008) Kent's Words of Estimative Probability. https://www.cia.gov/library

Manadhata PK, Wing JM (2011) An attack aurface metric. IEEE Trans Soft Eng 37(3):371–386 (ISSN 0098–5589)

Mauw S, Oostdijk M (2005) Foundations of attack trees. In: International conference on information security and cryptology ICISC 2005. LNCS, vol 3935. Springer, Berlin, pp 186–198

Morais A, Hwang I, Cavalli A, Martins E (2013) Generating attack scenarios for the system security validation. Netw Sci 2(3–4):69–80 (ISSN 2076–0310)

OMG (2007a) OMG systems modeling language (OMG SysML) specification. Object management group

OMG (2007b) OMG unified modeling language: superstructure 2.1.2. Object management group

Ouchani S (2014) Lenzini G (2014) Attacks generation by detecting attack surfaces. Procedia Comput Sci 32:529–536 [ISSN 1877–0509. The 5th international conference on ambient aystems, networks and technologies (ANT-2014)]

Sawilla R, Defence R&D Canada Ottawa (2007). Googling attack graphs. Technical memorandum. Defence R&D Canada-Ottawa

Sheyner OM (2004) Scenario graphs and attack graphs. PhD thesis, School of Computer Science. Pittsburgh, pp AAI3126929

Siveroni Igor, Zisman Andrea, Spanoudakis George (2010) A UML-based static verification framework for security. Requir Eng 15:95–118

Solhaug B, Seehusen F (2014) Model-driven risk analysis of evolving critical infrastructures. J Ambient Intell Humaniz Comput 5(2):187–204 (ISSN 1868–5137)

Symantec Corporation (2014) Internet security threat report-2014

Vijayakumar H, Jakka G, Rueda S, Schiffman J, Jaeger T (2012) Integrity walls: finding attack surfaces from mandatory access control policies. In: Proceedings of the 7th ACM symposium on information, computer and communications security (ASIACCS 12). ACM, pp 75–76