Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2014) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014)

Attacks Generation By Detecting Attack Surfaces

Samir Ouchani[a,*], Gabriele Lenzini[a]

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

Abstract

In the development process of software and systems, one of the main challenges is to detect the existing attack surfaces and their exact related attacks. In this paper, we address the issue of generating attacks related to systems that are designed by using SysML activity diagrams. For this purpose, we develop a practical framework to enable attack surfaces detection and producing their related attacks. First, we propose a formalization of SysML activity diagrams, then we rely on the standard catalog of attacks to build a library of attack patterns. The extracted patterns are modeled as SysML activity diagrams too. Finally, given an SysML activity diagram of a system and a library of SysML attack patterns, we propose an algorithm that automatically detects attacks surfaces relevant to a system under test and generate their related possible attacks.

Selection and peer-review under responsibility of Elhadi M. Shakshuki.

Keywords: Systems Attacks, Attack Patterns, Attack Surfaces, SysML, Activity Diagrams;

1. Introduction

There are two distinct yet related challenges in the development of reliable software and systems. One is about detecting errors at early as possible in the stage of the development's life-cycle. The other is assessing robustness to attacks, which requires precise measurements of the product and its design artifact's security levels and solutions to detect potential vulnerabilities. In this paper, we address this second challenge, and we propose a framework to detect *attack surfaces*[1] of software/systems modeled by using SysML activity diagrams.

From a modeling perspective in software and system engineering, SysML[2] is a prominent object-oriented graphical language which today became defacto standard for software and systems modeling. SysML reuses a subset of UML packages[3] and extends others with specific systems' engineering features such as probability. Mainly, it covers four perspectives of systems modeling: structure, behavior, requirement, and parametric diagrams. Particularly, SysML activity diagrams are behavioral diagrams used to model system behaviors at various levels of abstraction[4].

In order to detect attack surfaces, we first show how to model attack patterns and then we propose an algorithm that automatically discovers and generates all possible attacks that are specific to the system under test.

* This research is supported by FNR Luxembourg, project "Socio-Technical Analysis of Security and Trust", C11/IS/1183245, STAST.
Corresponding author. Tel.: +353-466-644-5743 ; fax: +353-466-644-35743.

E-mail address: samir.ouchani@uni.lu

Selection and peer-review under responsibility of Elhadi M. Shakshuki.

From a security perspective, a strong system[5] is one in which the cost of an attack is greater than the potential gain to the attacker. Conversely, a weak system is one where the cost of an attack is less than the potential gain. The cost of an attack should take into account not only money, but also time to recovery and potential punishment for criminal. An attack is an attempt to gain unauthorized access to a system services, resources, or information or the attempt to compromise the system integrity, availability, or confidentiality, as applicable. For this purpose different attacks modeling have been deployed such as: attack tree, attack graph, and network attack graph, are the most common. An attack tree is a tree where the root is the global goal, the leaves are attacks, and the internal nodes are actions realizing goals and sub-goal actions required to execute the attack. An attack graph represents the entire network state; its arcs represent state transitions caused by an attacker's actions. A network attack graph is composed of the computer network, the attacker, and the defender, and a state transition corresponds to a single action by the intruder, a defensive action by the system administrator, or a routine network action. Here, attack patterns are a generic representation designed to expose the exploited code flaws and to describe the common techniques, and logic that attackers use to exploit the system vulnerabilities.

Recently, many researches and organizations such as CAPEC[1] and WASC[2] show a special interest in attack patterns but, by using the existing attack modeling techniques, there is always a gap between the system and the attack semantics. To reduce this gap, we use SysML activity diagrams to model standard attacks proposed by CAPEC. The result is a standard library of potential attack templates. Precisely we propose to automatically generate attacks related to a given system modeled as SysML activity diagrams. These diagrams can call and communicate with other diagrams and allow for probabilistic behavior specification. To perform this step we need a formal representation of SysML diagrams.

This paper's contributions are the following: (a) we formalize SysML activity diagrams (Sec. 3), (b) we provide a library of attack patterns based on CAPEC catalog which we model in SysML (Sec.4). Then (c) we detect the existing attack surfaces in the system diagrams and (d) assign for each obtained attack surface a sub-set of the system-dependent attacks, precisely the potential attacks that can make significant damages on the system (Sec. 5).

2. Related Work

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

Attack modeling. Grunske and Joyce[6] propose a risk-based approach that creates modular attack trees for each component in the system. These trees are specified as parametric constraints, which allow quantifying the probability of security breaches that occur due to internal and external component vulnerabilities. Frigault and Wang[7] models probability metrics based on attack graphs as a special Bayesian network. Each node of the network represents vulnerabilities as well as the pre and post conditions. Jürjen. et al.[8] and Houmb et al[9] 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[10] extends UMLsec to model peer-to-peer applications along with their security aspects. It relies on the concept of abuse cases defined as UML use cases and state machine diagrams to represent attack scenarios. Anderson et. al.[11] generate attack scenarios from the threat model of the wireless security protocol. First, they collect attacks from vulnerabilities databases. Then, they classify them in terms of violated properties. Finally, they generate the protocol attack tree by relying to SecurelTree tool.

Attack surfaces detection. Gegick and Williams[12] 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. Heqing et. al.[13] distill 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.[14] 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.[15] identify the communication attack surfaces by considering intent-based attacks on application that do not hold

[1] http://capec.mitre.org, Common Attack Pattern Enumeration and Classification sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

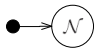
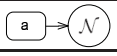
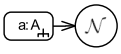
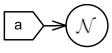
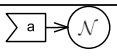
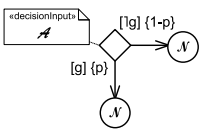
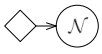
[2] http://www.webappsec.org, The Web Application Security Consortium.

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. [16] analyze 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 activity diagrams are graph-based models where vertices are activity nodes connected by activity edges. Nodes have three types: activity invocation, object and control nodes. Activity invocation includes receive and send signals (or objects), actions, and call behaviors. Activity control nodes can be initial, flow final, activity final, decision, merge, fork, and join nodes. Activity edges are of two types: control flow and object flow. Control flow edges are used to show the execution path through the activity diagram, and, object flow edges are used to show the flow of data between activity nodes. Concurrency and synchronization are modeled using forks and joins, whereas, branching is modeled using decision and merge nodes. While a decision node specifies a choice between different possible paths based on the evaluation of a guard condition (and/or a probability distribution), a fork node indicates the beginning of multiple parallel control threads. More especially, UML 2.0 activity forks model unrestricted parallelism. Thus, a token evolves asynchronously according to an interleaving semantics. Moreover, a merge node specifies a point from where different incoming control paths follow the same path, whereas a join node allows multiple parallel control threads to synchronize and rejoin. Table 1 describes the main artifacts that are used to design SysML activity diagrams and formalizes them which help to express easily our proposed framework.

Table 1. Formalization of SysML Activity Diagram Artifacts.

Artifacts	Formalization	Description
	$\bullet \rightarrowtail \mathcal{N}$	Initial node is activated when a diagram is invoked.
	$\odot$	Activity final node stops the diagram' execution.
	$\otimes$	Flow final node kills its related path' execution.
	$a \rightarrowtail \mathcal{N}$	Action node defines an atomic action.
	$a \uparrow \mathscr{A} \rightarrowtail \mathcal{N}$	Call behavior node invokes a new behavior.
	$a!v \rightarrowtail \mathcal{N}$	Send node is used to send a signal/object.
	$a?v \rightarrowtail \mathcal{N}$	Receive node is used to receive a signal/object.
	$D(\mathscr{A}, p, g, \mathcal{N}, \mathcal{N})$	Decision node with a call behavior $\mathscr{A}$, 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.

When a SysML activity diagram is invoked, its initial node is activated by a single token. The activation of any other node depends only on the deactivation of its preceded node and the guard satisfaction of its input edge. In addition,

the call behavior action or the decision node can consume its input token and invoke its specified behavior. In this case, the invocation supports both synchronous and asynchronous calls. In the asynchronous case, the execution of the invoked behavior proceeds without any further dependency on the execution of the activity containing the invoking artifact. But in the synchronous case, the execution of the calling artifact is blocked until it receives a reply token from the invoked behavior. For the case of decision nodes, when the invoked behavior enables more than one guard; the nondeterminism mechanism is adopted.

We present in Definition 1, the formal definition of SysML activity diagrams. Then, we derive two properties Property 1 and Property 2 that shape the structure and the control flow, respectively, of SysML activity diagrams.

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

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

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

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

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

4. Attack Patterns

In this section, we present the concept of attack patterns, introduce the notion of probabilistic likelihood of an attack, and model attack patterns as SysML activity diagrams.

The standard schema devised by the software assurance strategic initiative "CAPEC" includes both primary and supporting schema elements. Primary schema elements include the pattern id, the description of the attack, related weaknesses, typical severity, likelihood of exploitation, attack surface, and abstraction level. The supporting schema elements are categorized into describing, diagnosing, and enhancing information. Based on both schemes, we propose in Figure 1 the SysML activity diagram template of the attack patterns.

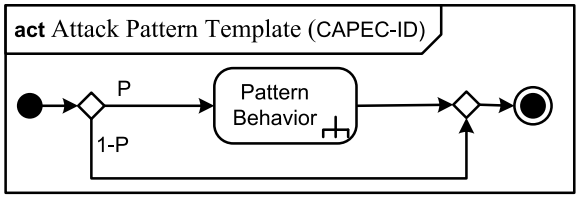


Fig. 1. SysML Activity Diagram of the Attack Pattern Template.

Each concrete attack pattern is built by instantiating this template and specifying the call behavior action denoted by "Pattern Behavior". The main control flow in this template is a probabilistic decision used to specify the likelihood of

the attack occurrence, which corresponds to the probability "P". The value of "P" is determined based on the "typical likelihood of exploitation" schema element provided within CAPEC catalog. However, this schema element is a qualitative description of the likelihood that ranges from "low" to "high". In order to quantify this attribute, we propose to assign ranges of probabilities to each qualitative description based on the standard of security risk management [17] in combination with the Kent's Words of Estimative Probability[3]. The probability ranges corresponding to the estimative terms are specified in Table 2. The probability related to the instantiated pattern is obtained by a centroid function.

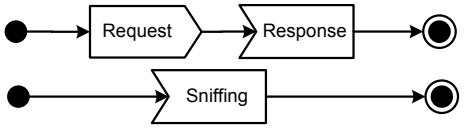
Table 2. Probability Values Scale.

CAPEC terms	Kent's Estimative terms	Probability values
High	Certain	100
High	Almost Certain	93% ($\pm$6%)
Medium to High	Probable	75% ($\pm$12%)
Medium	Chances About Even	50%($\pm$10%)
Low to Medium	Probably Not	30%($\pm$10%)
Low	Almost Certainly Not	7%($\pm$5%)
Low	Impossible	0

In the following, we provide three attack patterns with their corresponding activity flows representing the pattern behavior and their associated likelihoods.

- Spoofing (CAPEC-156): An attacker builds a message such that it is capable of masquerading an authorized message from a trusted principal. As a result, consumers of these messages can be manipulated into responding or processing the deceptive message. It may refer to spoofing the content (CAPEC-148) or the id (CAPEC-151). Their pattern is depicted by the following figure such that P(CAPEC-148)=P(CAPEC-151)=0.8.

- 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.



- 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 and launched by these probability values: P(CAPEC-125$|n \geq m$)= 0.8, P(CAPEC-131$|n < m$)=0.8 where n is the number of requests and m is a number fixed by the designer.

[3] https://www.cia.gov/library; Words of Estimative Probability.

5. Attacks Generation Framework

In this section, we detail our framework that automatically generates attacks related to a given system. The overview of our proposed framework is depicted in Figure 2. It takes a system modeled by a set of SysML activity diagrams as input. These diagrams can be obtained either by relying to the system specification document, or by reverse-engineering the system source code. To help generating the system-dependent attacks, the framework relies on the library of attack templates that are proposed in Section 4. Then, it provides an algorithm to detect attack surfaces from where an attack can damage the system. In addition, it 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. The result is a set of concrete attacks proper to the system under test.

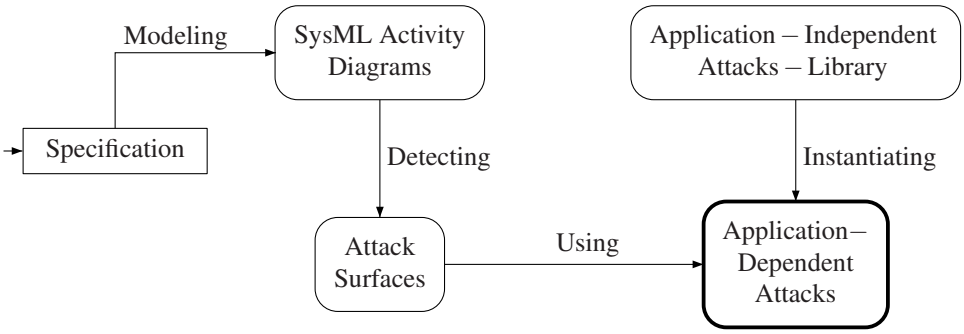


Fig. 2. System Attacks Generation Framework.

Attack Surfaces Detection

A system attack surface is the way in which an adversary can interact with the system and potentially cause damage to it. An attack surface is the subset of the system resources that an attacker can use to exploit the system. The relevant system resources that are part of the attack surface are: entry points, exit points, channels, untrusted data. An entry/exit points are artifacts to receive/send objects, respectively. A channel connects an attacker to a system, and, untrusted data is a persistent data which an entry/exit points can read (receive)/write (send), respectively. Hence, the larger the attack surface, the greater will be the number of potential attacks. The definition of an attack surface for SysML activity diagrams is given by Definition 2.

Definition 2 (Attack Surface). *An attack surface of $\mathscr{A}$ is the tuple $\omega = \langle N, X, O, Ch \rangle$, where:*

1. *N is the set of entry points that includes activity elements of type AcceptEventAction, AcceptTimeEventAction, InputPin, Action, ValuePin, ExpansionNode, and ActivityParameterNode.*
2. *X is the set of exit points that includes entities of kind SendSignalAction, SendObjectAction, OutputPin, Action, ExpansionNode, and ActivityParameterNode.*
3. *O is the set of untrusted data that includes ObjectNode, DataStoreNode, and CentralBufferNode.*
4. *$Ch : N \cup X \rightarrow O$ is a function assigning to each entry or exit point an of object that is denoted by ActivityEdge or ExceptionHandler.*

To obtain the set of attack surfaces of a SysML activity diagram $\mathscr{A}$, we parse the diagram with a depth-first search algorithm that is illustrated in Algorithm 1. The procedure Ξ takes $\mathscr{A}$ as input and produces a list of attack surfaces denoted by Ω. The procedure Ξ is described as follows. First, the initial node is pushed into the stack of nodes denoted

by *nodes* (line 8). While the stack is not empty (line 10-31), the algorithm pops a node from the stack *nodes* into the current node denoted by *cNode* (line 10). Then, the current node is added into the list *vNode* of visited nodes (line 12) if it is not already visited (line 12), and its successors saved in the list *nNode* (line 13). The explored successors are pushed into the stack *nodes* (line 27-29), then, they are cleared from the list *nNode* (line 30). Finally, the algorithm calls the function Λ (detailed in the next section) and it terminates since all nodes are visited. The complexity of the procedure Ξ is $\mathcal{O}(|n|^2)$ where $|n|$ is the number of activity nodes in $\mathscr{A}$. Our algorithm is sound by linking to each detected attack surface a concrete attacks from the library; in fact, we make an exhaustive search on all attack surfaces and we assign to them possible attacks which we take from the library. For the completeness, we argue that all the assigned attacks are the only possible ones for each detected attack surface, but we have not proven it yet.

Input: SysML activity diagrams $\mathscr{A}$.
Output: Attack Surfaces Ω.

1: *nodes* as **Stack**;
2: *cNode* as **Node**;
3: *nNode*, *vNode* as **list_of_Node**;
4: ω_1, ω_2 as **list_of_Nodes**;
5: ω_3 as **list_of_Objects**;
6: ω_4 as **list_of_Edges**;
7: **procedure** $\Xi(\mathscr{A})$
8: *nodes.push*(*in*);
9: **while** *not nodes.empty*() **do**
10: *cNode* := *nodes.pop*();
11: **if** *cNode* not in *vNode* **then**
12: *vNode.add(cNode)*;
13: *nNode* := *cNode.successors()*;
14: **if** *cNode.type()* in *N* **then**
15: $\omega_1.add(cNode)$;
16: **else**
17: **if** *cNode.type()* in *X* **then**
18: $\omega_2.add(cNode)$;
19: **else**
20: **if** *cNode.type()* in *O* **then**
21: $\omega_3.add(cNode)$;
22: **end if**
23: **end if**
24: $\omega_4.add(Edge(cNode, nNode)$;
25: **end if**
26: **end if**
27: **for all** *n in nNode* **do**
28: *nodes.push*(*n*);
29: **end for**
30: *nNode.clear();*
31: **end while**
32: $\Lambda(\omega_1 \cup \omega_2 \cup \omega_3 \cup \omega_4)$;
33: **end procedure**

Algorithm 1. Attack Surfaces Detection Algorithm.

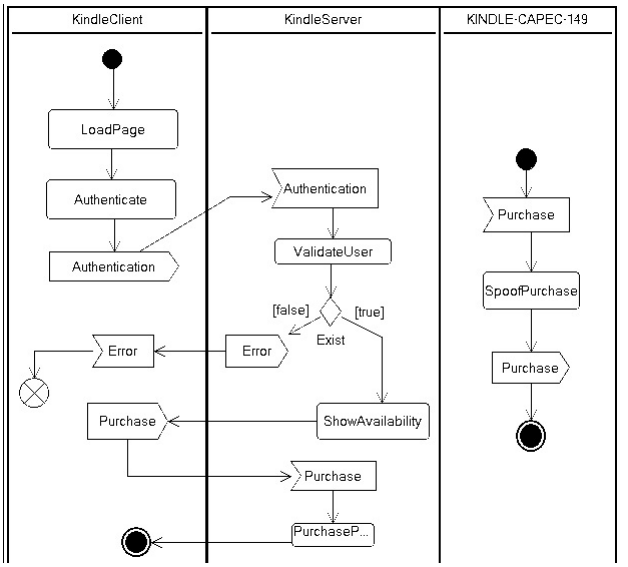


Fig. 3. The kindle diagram has three swimlanes (partitions): The kindle customer (left), the kindle server (middle) and the generated attack (right) for the "purchase" attack surface in the client side. Initially, the kindle customer loads the kindle online process, then authenticate by requesting authentication service via the kindle server, the server receives the authentication request in order to check the validity of the customer, next. If the authentication is valid, then the server shows the product availability and the kindle client sends his purchased items to be received by the server and processed later. Consequently, the server terminates the purchasing process. Otherwise, it sends the error message to the kindle client and the process ends after the delivery of the error message. The generated attack diagram presented in the right swimlane exploits the "purchase" attack surface in the client side and instantiates the template CAPEC-149. It launches attack on the client by receiving the purchase request, spoof the purchase content, and resend it to the kindle server. As a result, the purchase is diverted to the hacker account.

Application-Dependent Attacks Selection

Our objective is to assign the proper attack template for each attack surface obtained by Algorithm 1. Then, we instate this template to be dependent to the system under study. For that, we propose the function Λ in Listing 1

that assigns for each attack surfaces $\omega \in \Omega$ at least one attack $k \in \mathbb{K}$ such that $\mathbb{K} = \{k_1, k_2, k_3, ..., k_{10}\}$ where k_1 is CAPEC-148, k_2 is CAPEC-151, k_3 is CAPEC-116, k_4 is CAPEC-117, k_5 is CAPEC-125, k_6 is CAPEC-131, k_7 is CAPEC-148, k_8 is CAPEC-152, k_9 is CAPEC-225, and k_{10} is CAPEC-28. As example, we present in Figure 3 the generated attack related to the purchase attack surface in kindle online system.

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} ∪ Λ(N) end
4                    a ↣ N  ⇒ in {k2,k4} ∪ Λ(N) end
5                    a ↑ A ↣ N  ⇒ in {k3,k4,k10} ∪ Λ(N) ∪ Λ(A) end
6                    a!v ↣ N  ⇒ in {k2,k3,k4} ∪ Λ(N) end
7                    a?v ↣ N  ⇒ in {k1,k8} ∪ Λ(N) end
8                    D(A,p,g,N1,N2) ⇒ in {k5,k7,k9} ∪ Λ(N) ∪ Λ(A) end
9                    M(x,y) ↣ N  ⇒ in {k5,k9} ∪ Λ(N) end
10                   F(N1,N2) ⇒ in {k5,k9} ∪ Λ(N) end
11                   J(x,y) ↣ N  ⇒ in {k5,k9} ∪ Λ(N) end
```

6. Conclusion

In this paper, we presented an attack generation framework based on detecting attack surfaces of a system modeled by using SysML activity diagrams. To generate attacks, we developed a library of system attacks, and we devised an algorithm that detects attack surfaces of the system. Further, a function is proposed to assign a set of attacks to the detected attack surfaces. The presented work can be extended in the following directions. We intend to generate attack surfaces related to real time systems, and cover more attacks such as the social engineering attacks. In addition, we want to formally prove the soundness and completeness of our framework, and validate it on specific use cases, such as protocols for mobile authentication, and authenticated m-banking transactions which are scenarios we have in strategic projects of our institution.

References

1. Manadhata, P., Wing, J.. An Attack Surface Metric. *IEEE Trans on Soft Eng* 2011;**37**(3):371–386.
2. AA.VV., . *OMG Systems Modeling Language (OMG SysML) Specification*. Object Management Group; 2007.
3. AA.VV., . *OMG Unified Modeling Language: Superstructure 2.1.2*. Object Management Group; 2007.
4. Holt, J., Perry, S.. *SysML for Systems Engineering*. Institution of Engineering and Technology Press; 2007.
5. OGorman, L.. Comparing Passwords, Tokens, and Biometrics for User Authentication. *Proc of the IEEE* 2003;**91**(12):2021–2040.
6. Grunske, L., Joyce, D.. Quantitative Risk-Based Security Prediction for Component-Based Systems with Explicitly Modeled Attack Profiles. *J Syst Softw* 2008;**81**:1327–1345.
7. Frigault, M., Wang, L.. Measuring Network Security Using Bayesian Network-Based Attack Graphs. In: *Proc. of the 32nd IEEE Int. Computer Software and Applications Conf. (COMPSAC '08)*. 2008, p. 698–703.
8. Jürjens, J., Shabalin, P.. Automated Verification of UMLsec Models for Security Requirements. In: *UML 2004 The Unified Modeling Language, volume 2460 of LNCS*. Springer; 2004, p. 412–425.
9. Houmb, S.H., Islam, S., Knauss, E., Jürjens, J., Schneider, K.. Eliciting Security Requirements and Tracing them to Design: An Integration of Common Criteria, Heuristics, and UMLsec. *Requir Eng* 2010;**15**:63–93.
10. Siveroni, I., Zisman, A., Spanoudakis, G.. A UML-Based Static Verification Framework for Security. *Requir Eng* 2010;**15**:95–118.
11. Morais, A., Hwang, I., Cavalli, A., Martins, E.. Generating attack scenarios for the system security validation. *Networking Science* 2013; **2**(3-4):69–80.
12. Gegick, M., Williams, L.. On The Design of More Secure Software-Intensive Systems by Use of Attack Patterns. *Inf Softw Technol* 2007; **49**:381–397.
13. Huang, H., Zhang, S., Ou, X., Prakash, A., Sakallah, K.A.. Distilling critical attack graph surface iteratively through minimum-cost sat solving. In: *ACSAC'11*. 2011, p. 31–40.
14. Vijayakumar, H., Jakka, G., Rueda, S., Schiffman, J., Jaeger, T.. Integrity walls: Finding attack surfaces from mandatory access control policies. In: *Proc. of the 7th ACM Symp. on Information, Computer and Communications Security (ASIACCS 12)*. ACM; 2012, p. 75–76.
15. Kantola, D., Chin, E., He, W., Wagner, D.. Reducing attack surfaces for intra-application communication in android. In: *Proc. of the 2nd ACM Work. on Security and Privacy in Smartphones and Mobile Devices (SPSM 12)*. ACM; 2012, p. 69–80.
16. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., et al. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In: *Proc. of the 20th USENIX Conference on Security (SEC 11)*. USENIX Association; 2011, p. 6–6.
17. *Information technology, Security techniques, Information security risk management*. Int. Organization for Standardization (ISO); 2008.