# SysML-Sec Attack Graphs: Compact Representations for Complex Attacks

Ludovic Apvrille[1](✉) and Yves Roudier[2]

[1] Institut Mines-Telecom, Telecom ParisTech, CNRS LTCI, Sophia Antipolis, France
ludovic.apvrille@telecom-paristech.fr
[2] EURECOM, Sophia Antipolis, France
yves.roudier@eurecom.fr

**Abstract.** We discuss in this paper the use of SysML-Sec attack graphs as a graphical and semi-formal representation for complex attacks. We illustrate this on a PC and mobile malware example. We furthermore provide examples of the expressivity of the operators used in such diagrams. We finally formalize the attack traces described by these operators based on timed automata.

## 1 Introduction

Modeling security threats in distributed systems, and even more so in embedded system is a usual aspect of the work of security analysts. However, more than often, the threat analysis simply relies on the knowledge of specific malware and their variants, or on the exploitation of well-known vulnerabilities rather than in finding new combinations of attacks.

Unfortunately, an increasing number of embedded systems have become communicating artifacts, feature new interactions with their immediate environment or with backend systems, and are thus exposed to criminals. Many of these security issues reflect either the exploitation of low-level vulnerabilities, which might often be addressed with appropriate programming practices and specific component tests, or design flaws due to an insufficient understanding of the mapping of functional or security logical components to the hardware architecture.

We introduced in the SysML-Sec framework [2] a more systematic representation of attacks envisioned or known to be feasible on the system under design and/or development. In the framework of the activities undertaken when following a Model-Driven Engineering (MDE) approach, the attack modeling phase is known as a very important driver for motivating the need for introducing security countermeasures in a risk analysis, and also for selecting where those security mechanisms better fit.

SysML-Sec extends SysML's parametric diagram in order to depict attacks, their composition, and to represent the assets target of these attacks in an attack graph. We also discuss in this paper the use of attack graphs and their operators and define their formal semantics based on timed automata (novel contribution). We also introduce a more complete example of application of such an analysis to model the Zeus/Zitmo mobile malware that were not published before.

S. Mauw et al. (Eds.): GraMSec 2015, LNCS 9390, pp. 35–49, 2016.
DOI: 10.1007/978-3-319-29968-6_3

## 2 Attack Modeling

**Threats and Attacks.** Threats and security vulnerabilities of the selected assets should as much as possible describe the capabilities that an attacker should meet or exceed and the origin of attacks (local, remote, through a specific interface). The SysML-Sec environment supports the assessment of risks following the approach described in more detail in the EVITA case study [8,13]. We also implemented automated checks of the threat coverage by security objectives. Based on the risk analysis, one should also identify and prioritize security objectives that are mapped to a threat.

**Attack Graphs.** Instead of using the traditional attack tree approach [14], we suggest that threats can be better modeled with a more relational approach, using slightly customized SysML Parametric Diagrams. Threats are modeled as values embedded into blocks representing the target of the attacks, thus achieving a representation that visually emphasizes the assets. Attacks ($<< attack >>$ stereotype) can be linked together with a few primitive operators. Those operators are either logical operators like $AND$, $OR$, and $XOR$, or temporal causality operators like $SEQUENCE$, $BEFORE$, or $AFTER$. We consider the latter constructs as especially helpful to describe the attacker's operational point of view in embedded systems, like for instance situations in which there is a maximum duration between two causally related attacks. For example, when attacking a system with time-limited authentication tokens, the token must be first retrieved, and then the use of this token must occur before its expiration.

Attack instances in different parametric diagrams can be linked together in order to assess the impact of a specific vulnerability and the need to address it at the risk assessment phase. An attack can also be tagged as a *root* attack, meaning that this attack is at the top of a tree of attacks. In other words, such an attack is not used to built up more complex attacks. Last but not least, attacks can be linked to requirements, thus allowing an automated check of the coverage of attacks by verifying whether each attack is linked to at least one security requirement.

The attacks in multiple diagrams finally result in a directed graph whose vertices can be either individual attacks (or leaf attacks), intermediate attacks (resulting from the composition of multiple other attacks), or operators that combine other attacks. We currently only consider acyclic graphs, but we are currently considering an extension to cyclic graphs in order to model resource usage (see the discussion in Sect. 6). Last but not least, we do not claim that these operators are always well adapted for modeling attack graphs, but at least, attack graphs offers a richer semantics than the one of attack trees, thus leading to more compact representations (in other words: less operators must be used). Also, attacks graphs demonstrated their ability to model complex attack scenarios, e.g. Zeus/Zitmo.

## 3 Example: Modeling Zeus/Zitmo

The Mobile component of the ZeuS crimeware kit (also known as or Trojan-Spy.*.Zitmo) was released in 2010 in order to intercept mobile Transaction Authentication Numbers (*mTAN* codes) from mobile phones.

The PC/Windows component, Zeus, modifies the browser of Microsoft Windows computers with a malicious plugin, so that any attempt to access an online bank website redirects the request to a fake bank site provided by the attacker. Additionaly, a keylogger spies username/password pairs to make it possible for the attacker to log undetected into the real banking system of the user. Zitmo also maliciously suggests the user to install a fake mobile bank application on his/her mobile phone. Once done, the fake application spies received SMS messages so as to silently steal mTANs.

The SysML-Sec attack graph of this trojan is given in Fig. 1. It has been made with TTool [1]. The system attacker is modeled with two main sub-blocks: the attacker PC that is used to gather information on users credentials (username,

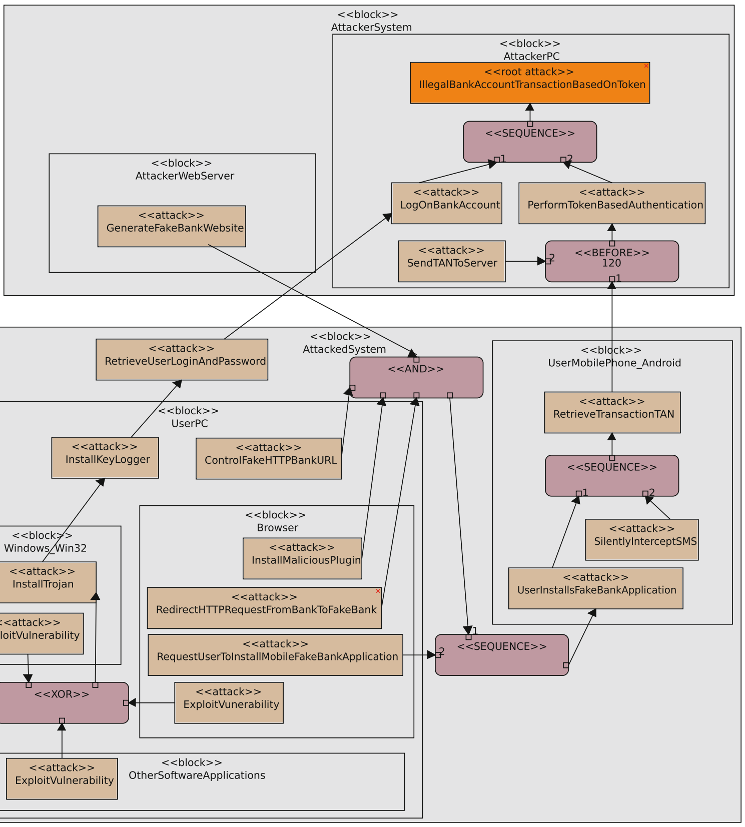


**Fig. 1.** Zeus/Zitmo attack graph (model made with TTool)

password, mTAN) and to perform illegal transactions using those credentials, and a webserver used to host fake bank websites. The attacked system consists in both the Windows PC of the targeted person, and his/her Android mobile phone. The first exploit is performed on the Windows PC, either using a Win32 exploit, or a browser exploit, or using other exploits in applications: the attack graph model thus contains three sub-blocks in the "UserPC" block. The XOR operator expresses that as soon as one exploit was performed on the targeted PC, the trojan can be installed and no further exploit linked to the XOR is useful. The trojan intercepts the username and password of the user, and sends them back to the attacker system. In parallel, several attacks are necessary in order to intercept requests to the bank system: the attacker must settle a fake bank server. The attacker must also control the *http* request to the bank system. He/She also has to install a malicious plugin in the browser of the attacked PC. Once all this has been done (AND operator), the browser can ask the user to install a fake Android application on his/her mobile phone (SEQUENCE operator in the bottom right part of the model). Once installed, the fake application can silently monitor SMS (SEQUENCE operator in the "UserMobilePhone_Android" block), and thus retrieve *mTANs*. When an mTAN has been obtained, the attacker has 120 seconds to use it (BEFORE operator).

## 4 Semantics of Attack Graph Constructs

The semantics of the attack traces are captured by a timed automaton which is the result of the parallel and synchronized composition of the automata expressing the potential occurrences and re-occurrences of individual attacks together with automata expressing the behavior of the operators that describe how these attacks are composed. Without any loss of generality, we depict in the following the automata generated by a binary combination of two attacks (but they support more than two attacks).

Individual attacks, which would be the leaves of an attack tree, can be modeled as depicted for *attack1* in Fig. 2. An attack can:

- Succeed ($a1!$). In that case, it can be performed again afterwards.
- Be stopped ($stop\_a1?$). An attack is stopped when the system does not allow the activation of such an attack after all related automata of the attack graph are synchronized, e.g., an XOR operator forbids the execution of that attack.

### 4.1 Intermediate Attacks

Intermediate attack nodes in the graph play an important role in the composition of attacks, and as such, interconnecting operators. Such a node corresponds to the success of one or more attacks that precedes it in the directed attack graph according to the semantics of the preceding operator. The semantics of those nodes must more specifically support the backward propagation of *stop* events within the graph. Thus, an intermediate attack (see Fig. 3) first

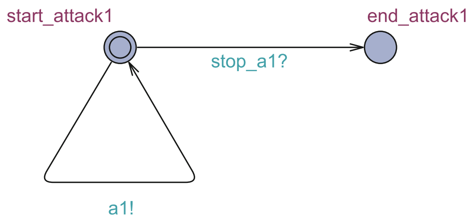


**Fig. 2.** Semantics of an individual attack

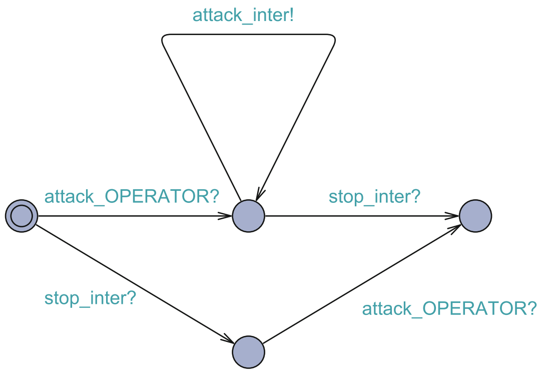


**Fig. 3.** Semantics of an intermediate attack node

waits for its activation operator (*attack_OPERATOR*), then, it can be executed several times (*attack_inter*), or be stopped (*stop_inter*). Also, before its activation operator is complete, it can be stopped ((*stop_inter* from the initial state): in that latter case, only the completion of its operator can be performed (*attack_OPERATOR*).

Finally, we assume that an oriented connection between attacks *attack1* to *attack2* is a shortcut for *attack1* to an OR node, and then from the OR node to *attack2*.

## 4.2 And Operator

The AND operator models the expectation that multiple attacks are required to be executed in conjunction (possibly in a parallel fashion). Failing to achieve any of the elementary attacks results in the overall failure of subsequent dependent attacks. For instance, many malware rely on checks to make sure they are not running in a virtualized honeypot: all those checks should succeed and thus can be modelled as attacks under an AND.

The timed automaton formalizing the behavior of the operator is depicted in Fig. 4. It performs the synchronization of the automata of the underlying attacks. The handling of an additional attack would result in an additional transition at the second state of this timed automaton.

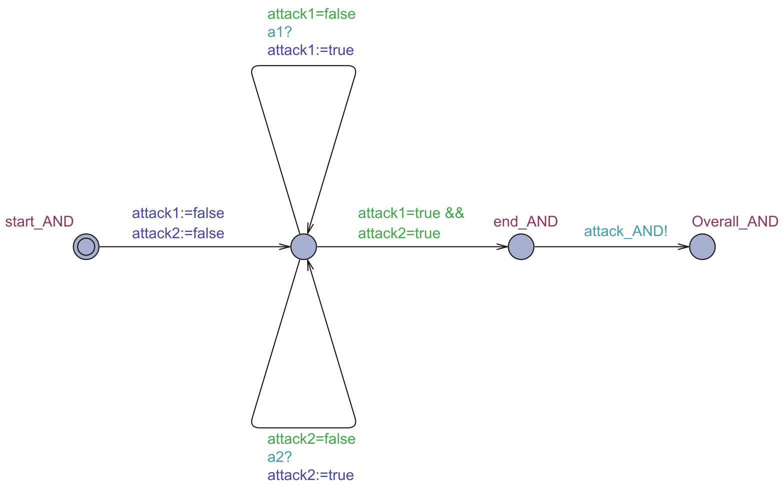


**Fig. 4.** AND operator

## 4.3 Or Operator

The OR operator models a situation in which multiple attacks can be executed to enable other composite attacks. The first successful attack will enable the execution of new composite attacks farther in the attack graph. Also not all attacks under the OR operator need to be performed before a composite attack using the OR proceeds or even succeeds (see Fig. 5).

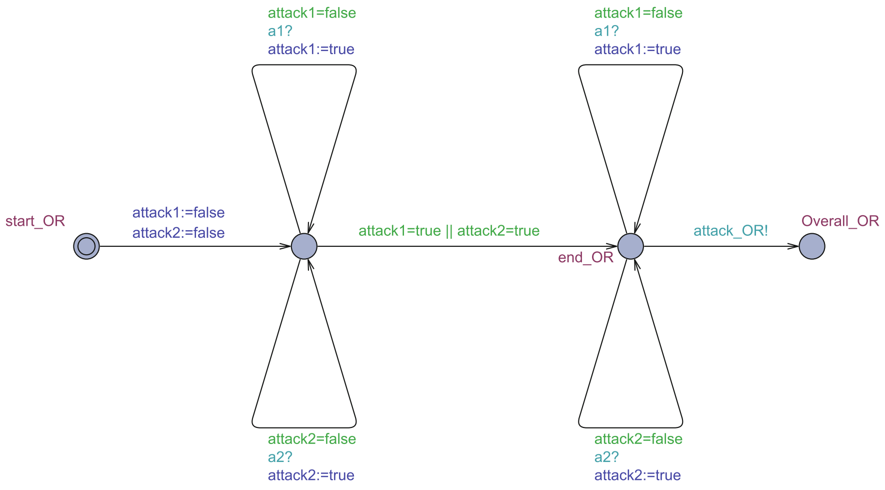


**Fig. 5.** OR operator

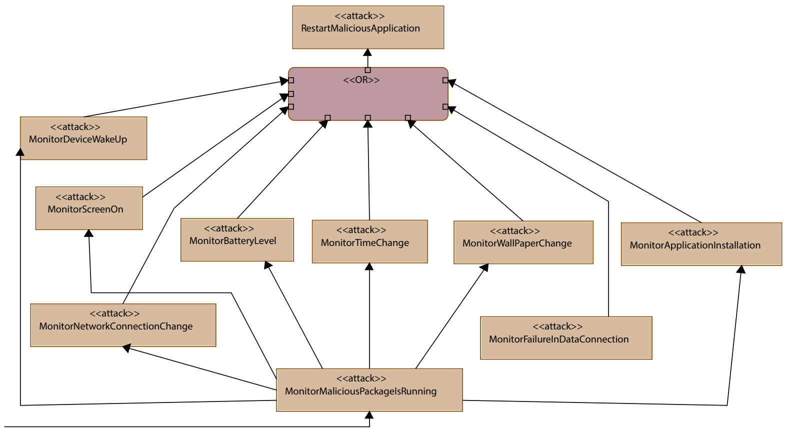

<<attack>> RestartMaliciousApplication
<<OR>>
<<attack>> MonitorDeviceWakeUp
<<attack>> MonitorScreenOn
<<attack>> MonitorBatteryLevel
<<attack>> MonitorTimeChange
<<attack>> MonitorWallPaperChange
<<attack>> MonitorApplicationInstallation
<<attack>> MonitorNetworkConnectionChange
<<attack>> MonitorFailureInDataConnection
<<attack>> MonitorMaliciousPackageIsRunning

**Fig. 6.** OR operator - Excerpt of the attack graph of Chuli

This operator can for instance model redundant operations that an attacker or a malware may perform for instance to extract some information.

Let's take the example of an OR operator taken from the model of the Chuli Android mobile malware [7]. Basically, Chuli infects mobile phones through spam emails, and then sends to the remote attacker's server private information contained on the mobile phone. One interesting feature of this malware is its ability to monitor whether it is running or not using callback services triggered by external events, e.g., *ScreenOn* and *BatteryLevel* events. As soon as one of this event occurs in the system, Chuli can restart its main application, if necessary. Thus, all those trigger events can be monitored in parallel. Said differently, one among all events is enough for Chuli to perform the check. Also, once one event has been used by Chuli, Chuli continues other events. All this corresponds to an OR operator, see Fig. 6.

## 4.4 XOR Operator

The XOR operator models alternative and exclusive independent attacks. Thus, the behavior of interest expressed by this operator is the success of a single attack. Said differently, any first successful attack among those referenced by the operator is the one that will appear in the trace of the attack at the exclusion of all others.

The semantics with OR is different because an XOR forbids the execution of other attacks, apart form the first successful one. On the contrary, OR does not impose any constraint on other attacks. For example, in a situation in which attacks are tested in parallel - for example, a monitor waiting for several callbacks informing about a success -, then the OR operator shall be used. In a situation where only one of the attack is tested, one after the other, without imposing the order of testing, then, the XOR operator shall be used.

More formally (see Fig. 7), once one attacks has been successfully performed ($a1?$ or $a2?$), the attack that was not performed is deactivated ($stop\_a1!$ or $stop\_a2!$), and then the intermediate attack is executed ($attack\_XOR$).

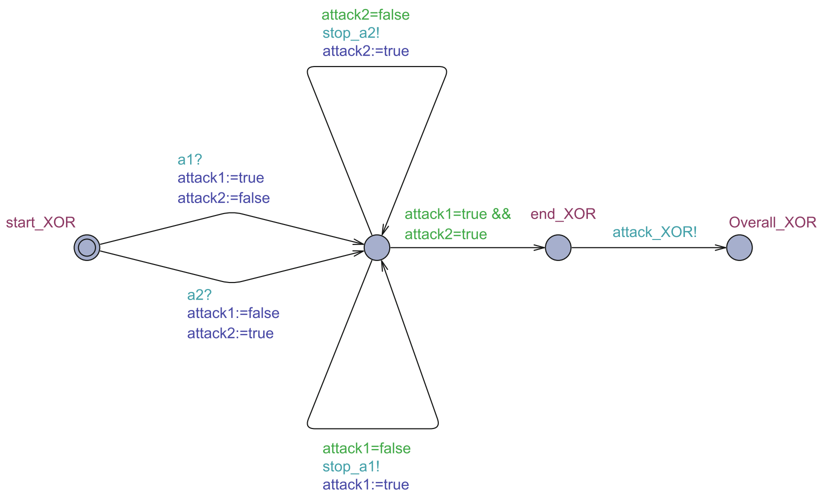


**Fig. 7.** XOR operator

## 4.5 SEQUENCE Operator

The SEQUENCE operator models attacks which must be performed in a strict order $a_1, a_2, ...; a_n$ (see Fig. 8). Failing to achieve one attack $a_i$ makes it impossible to subsequently execute attacks $a_j$ with $j > i$.

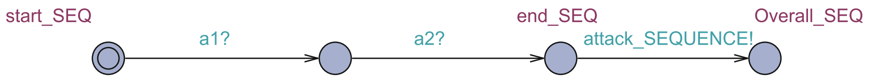


**Fig. 8.** Sequence operator

## 4.6 BEFORE Operator

The BEFORE operators is based on a sequence of attacks with a **maximum** duration between two consecutive attacks (see Fig. 9). Just like for the SEQUENCE, failing to achieve one attacks makes it impossible to achieve subsequent attacks. Moreover, failing to achieve one attack within its given allowed period of execution also makes it impossible to execute subsequent attacks.

This operator is particularly suited to model life-time limited tokens.

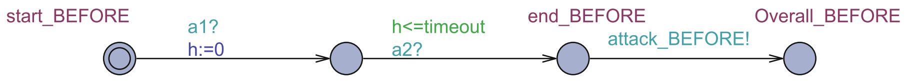


**Fig. 9.** Before operator

### 4.7 AFTER Operator

The AFTER operators is based on a sequence of attacks with a **minimum** duration between two consecutive attacks (see Fig. 10). Just like for the SEQUENCE, failing to achieve one attacks makes it impossible to achieve subsequent attacks. Moreover, if an attack is available for execution before the minimum duration, the system will force it to execute only after the minimum duration.

The AFTER operator is particularly interesting to model situations in which an attack is useless before waiting for an access to be available, e.g., when brute-forcing a password system with a minimum delay between two attempts.

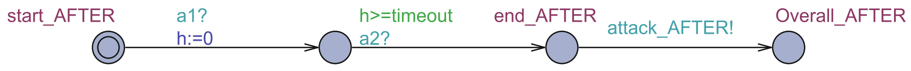


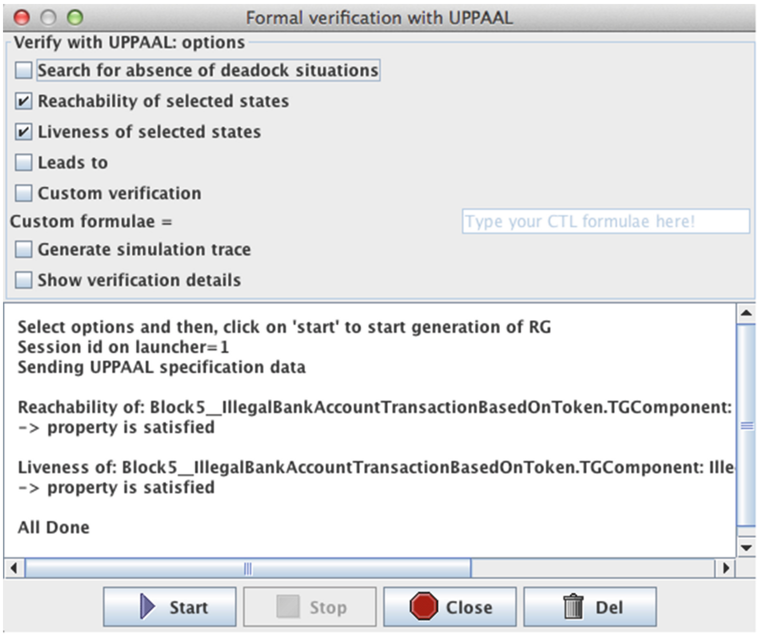
**Fig. 10.** After operator

## 5 System Validation

From a formal verification perspective, attack graphs can be formally analyzed directly from TTool, in terms of reachability, liveness and "leads to" properties on attacks.

- Reachability of an attack $a$. Means that there exists at least one possible series of attacks $a_1, a_2, ..., a_n, a$ (i.e., trace of attacks) that leads to $a$.
- Liveness of an attack $a$. Means that whatever the possible traces of attacks in the system $a_1, a_2, ...; a_n$, $\exists i / a_i = a$.
- Liveness of an attack $b$ after another attack $a$ was performed. Means that whenever a trace of attacks contains $a = a_i : a_1, a_2, ...; a_n$, $\exists j > i / a_j = b$. This property is commonly named "leads to" (this is the case in TTool) or also "response".

From SysML-Sec models edited in TTool, a user can either simulate the model, or perform formal proofs with UPPAAL [3]. The simulation engine integrated in TTool allows usual commands (step-by-step execution, reaching next breakpoint, etc.), and animates the attack graph while it is simulated. A sequence diagram representing the trace of performed attacks is displayed as well. Formal proofs can also be performed with a press-button approach directly from TTool (but UPPAAL needs to be installed): indeed, TTool automatically transforms



Verify with UPPAAL: options

- [ ] Search for absence of deadock situations
- [✓] Reachability of selected states
- [✓] Liveness of selected states
- [ ] Leads to
- [ ] Custom verification

Custom formulae = Type your CTL formulae here!

- [ ] Generate simulation trace
- [ ] Show verification details

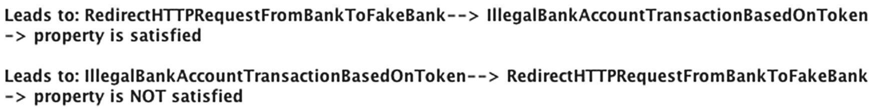
```
Select options and then, click on 'start' to start generation of RG
Session id on launcher=1
Sending UPPAAL specification data

Reachability of: Block5__IllegalBankAccountTransactionBasedOnToken.TGComponent:
-> property is satisfied

Liveness of: Block5__IllegalBankAccountTransactionBasedOnToken.TGComponent: Ille
-> property is satisfied

All Done
```

**Fig. 11.** Reachability and liveness of the main attack (TTool dialog window)



```
Leads to: RedirectHTTPRequestFromBankToFakeBank--> IllegalBankAccountTransactionBasedOnToken
-> property is satisfied

Leads to: IllegalBankAccountTransactionBasedOnToken--> RedirectHTTPRequestFromBankToFakeBank
-> property is NOT satisfied
```

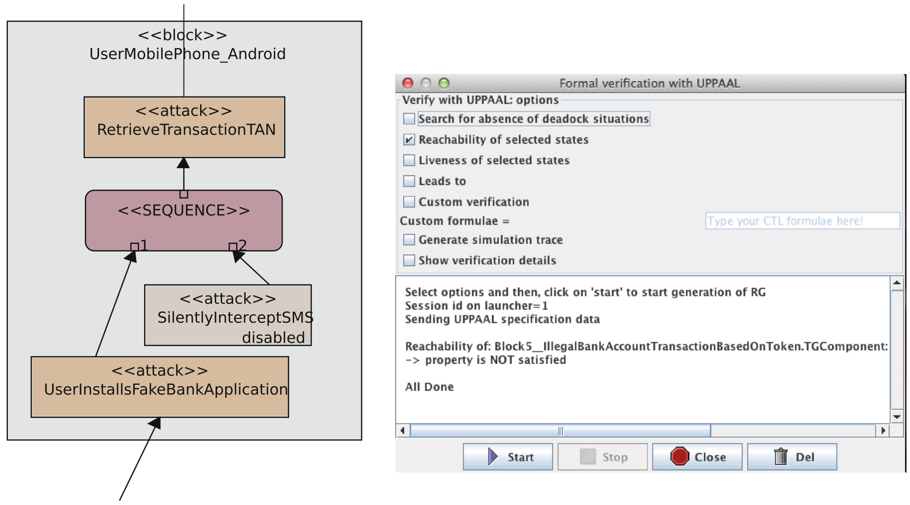
**Fig. 12.** "Leads to" property proved from TTool

the attack graphs into a UPPAAL specification, feeds it into UPPAAL, gets the results, and presents them in a friendly way. This model transformation is instantaneous from a user's perspective in all case studies we've made (linear algorithm). The formal proof complexity obviously depends on the model concurrency, e.g., the use of OR operators increases the concurrency, whereas the use of SEQUENCE constraints traces.

Figure 11 displays the reachability and liveness dialog window of TTool for the "root attack" ("IllegalBankAccountTransactionBasedOnToken") of the Zitmo model (Fig. 1). Both the reachability and liveness are satisfied.

A "leads to" property can be evaluated if two attacks have been selected. For instance, in the Zitmo model (Fig. 1), we can select the two attacks $a_1$ = "RedirectHttpRequestFromBankToFakeBank" and $a_2$ = "IllegalBankAccountransactionBasedOnToken" (see Fig. 12): The "leads to" property holds for $a_1 \leadsto a_2$ but not for $a_2 \leadsto a_1$.

TTool also allows to enable/disable attacks of attacks trees, so as to understand what is the importance/impact of an attack on the system. For example,



Verify with UPPAAL: options
☐ Search for absence of deadock situations
☑ Reachability of selected states
☐ Liveness of selected states
☐ Leads to
☐ Custom verification
Custom formulae = Type your CTL formulae here!
☐ Generate simulation trace
☐ Show verification details

```
Select options and then, click on 'start' to start generation of RG
Session id on launcher=1
Sending UPPAAL specification data

Reachability of: Block5__IllegalBankAccountTransactionBasedOnToken.TGComponent:
-> property is NOT satisfied

All Done
```

**Fig. 13.** An attack has been disabled in the Zitmo attack graph (Diagram on the left). Because of the disabled attack, the root attack cannot be performed anymore (right part of the figure).

if we disable the attack "SilentlyInterceptSMS" (left part of Fig. 13), then, the root attack is not reachable anymore (right part of Fig. 13).

## 6 Combining Operators and Attacks

This section discusses ways to handle complex attack relations relying on the relations between attacks described in Sect. 4.

### 6.1 Prioritizing Attacks Under a XOR

The XOR operator imposes no priority on the execution of the possible attacks. However, such an order may be achieved by combining an XOR with all the acceptable orderings of individual attacks, as can be described using the SEQ operator. Such a composite operator can be implemented based on the operators described above but requires generating all possible interleavings. To simplify the specification, we suggest the definition of a macro operator, SXOR. Such a macro operator could be integrated in the TTool environment. In the longer term, if such operators would prove useful, they may be standardized as a library shared by all SysML-Sec designers.

### 6.2 Compatibility Between Temporal Constraints

The joint use of AFTER and BEFORE can lead to situations where attacks are not reachable, because of the timing values of these operators. For example,

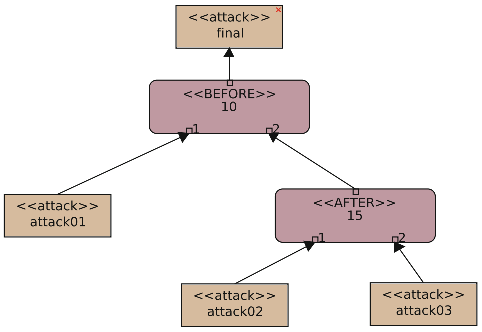


**Fig. 14.** The "final" attack cannot be performed because the two temporal constraints are not compatible

in Fig. 14, the root attack is not reachable because an attack is required to be performed before 10 units of time. But the AFTER operator forbids that situation. Modifying the temporal value in AFTER and BEFORE can make the root attack reachable, for example, by using the same temporal value. TTool can already analyze such situations, i.e., it can identify non reachable attacks because of non compatible timing constraints.

### 6.3 Cycles and Reachability

Cycles can be obtained in attack graphs by linking an attack generated from an operator to operators that were already handled previously in the trace of attacks.

For example, if we consider Fig. 15, *rootattack2* is reachable because the cycle occurs on a OR operator. If the same cycle is performed on the AND operator, then, the latter can never be executed, and so, *rootattack1* is not reachable.

Currently, such a situation is not supported by TTool, and by our semantics. A finer control over the use of cycles in general will require defining how many executions of the same operator can be allowed, for instance by adding

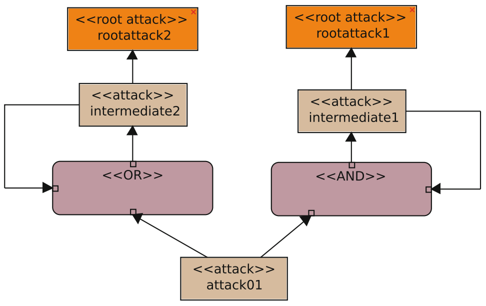


**Fig. 15.** Cycles. rootattack2 is reachable, but not rootattack1

an execution counter on each operator. Similarly, as long as they have not been explicitly stopped, attacks can be performed an infinite number of times: counters should also be added to these constructs. This work should be especially useful in the face of the modeling of denial of service attacks, which require not only a qualitative, but a quantitative assessment of the attacker's capabilities. We plan to develop these techniques as part of our future work.

## 7 Related Work and Perspectives

The formalism of attack trees brought to light by [14] has long been used to describe threats to applications and systems, and attacks to implement those threats. In that respect, attack trees are closely related to fault trees in dependable computing. Attack trees follow a goal-oriented approach that matches the objectives of an attacker and roughly describes an attack trace. However they capture a unique trace, and make it hard capturing complex attack scenarios built upon sub-attacks. They also fail at capturing the architectural components involved in a given attack with regards to the assets under attack, even though this often constitutes an important information for the trustfulness one can put into a component. In our case, location of attacks are given by their mapping onto architectural components.

Multiple variants of attack trees have been developed: they introduced operators with increasingly advanced semantics, e.g., [10], yet that have not addressed the above-mentioned issues. Our work tries to address these concerns based on the structure of our attack graphs rather than based only on the operators themselves. Among other benefits, this structure simplifies the reuse of sub-attacks without any duplication.

Attack graphs have been proposed and formalized even before attack trees received a widespread audience, like for instance privilege graphs [5], and more recently in order to automatically generate them from other formalisms [15]. [5] particular emphasized the quantitative aspect of the security assessment of threats. A Markovian model was used to determine the privileged edges in an attack graph. Our work also aims to introduce quantitative assessments while still retaining the hierarchical modeling that made the success of attack trees, and which is also connected with the system architecture in SysML-Sec in contrast with the "maze" graph described by the authors of [5].

Extensions were suggested to complement the static attack tree representations with more dynamic models. For instance, Petri net based approaches [6,12] were proposed in order to describe the triggering of different phases of an attack within an attack tree. [11] also suggested the use of Markovian processes (BDMP) to describe relationships between different attacks organized in a tree-like fashion but whose triggering could be independent from that structure. More recently, [16] relies on attack trees to complement the static analysis and dynamic analysis of Android malware: Nodes are enriched with e.g., permissions and capabilities ("P": Possible to realize; "I": impossible to realize). Other formalisms than attack trees have been introduced in order to capture attacks, but they are generally targeting security mechanisms first. We can mention modeling environments such as UMLSec [9], and tools for the proof of security properties in security protocols [4].

In a way, all these models also describe attack graphs with edges corresponding to different relationships. However, the approach described in this paper mostly focuses on expressing multiple attack traces. It aims at understanding whether a system is vulnerable and thus help deciding which security counter-measures might be most important through attack reachability and liveness analyses. Indeed, TTool facilitates the activation/deactivation of attacks in the graph, thus allowing to analyze the reachability and liveness of attacks in different situations. Combined with the location of attacks, this helps determining which and where attacks should be addressed first. We also believe that the modeling of our phases is more straightforward than the approaches we just outlined, because it is more rich w.r.t. attack trees, and more prone to the modular expression of threats due to the asset-centric distribution of attacks.

## 8 Conclusion and Future Work

From our experience, partitioning is a very important element when modeling attacks in order to understand both the assets at risk, their potential vulnerabilities, as well as the capabilities of the attacker. Thus, SysML-Sec proposes to use iterations between security requirements, attack graphs and partitioning models. Attack graphs adopt a block-centric perspective with reuse in mind. We especially think that this will allow for the composition of the threat modeling performed by security analysts about components over-the-shelf (COTS) with system specific analyses.

A few extensions of our work have already been discussed in Sect. 6. We plan to further extend SysML-Sec expressivity as follows: our declarative approach should be especially useful in order to incorporate knowledge from other threat modeling approaches. In that respect, our proposal explicitly maps attacks to the architecture, and makes it possible to introduce an abstract model of the attacker within the SysML parametric diagram for threat modeling. We essentially plan to extend our approach towards more quantitative assessments of threats, and also to integrate together attack graphs and risk assessment, e.g., using risk values on edges between attacks and operators.

## References

1. Apvrille, L.: TTool website (2013). http://ttool.telecom-paristech.fr/
2. Apvrille, L., Roudier, Y.: SysML-sec: a sysML environment for the design and development of secure embedded systems. In: APCOSEC , Asia-Pacific Council on Systems Engineering, 8–11 September 2013, Yokohama, Japan. Yokohama, JAPAN (09 2013) (2013). http://www.eurecom.fr/publication/4186
3. Bengtsson, J.E., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)

4. Blanchet, B.: Automatic verification of correspondences for security protocols. J. Comput. Secur. **17**(4), 363–434 (2009)
5. Dacier, M., Deswarte, Y., Kaâniche, M.: Information systems security, pp. 177–186. Chapman & Hall Ltd, London, UK (1996). http://dl.acm.org/citation.cfm?id=265514.265530
6. Dalton, G., Mills, R., Colombi, J., Raines, R.: Analyzing attack trees using generalized stochastic petri nets. In: Information Assurance Workshop, 2006 IEEE, pp. 116–123, June 2006
7. Fortinet: The Android/Chuli.A!tr.spy virus, March 2013. http://www.fortiguard.com/encyclopedia/virus/#id=4805535
8. Henniger, O., Apvrille, L., Fuchs, A., Roudier, Y., Ruddle, A., Weyl, B.: Security requirements for automotive on-board networks. In: ITST, Lille, France (2009)
9. Jürjens, J.: UMLsec: extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002)
10. Khand, P.: System level security modeling using attack trees. In: 2nd International Conference on Computer, Control and Communication, 2009. IC4 2009, pp. 1–6, February 2009
11. Piètre-Cambacédès, L., Bouissou, M.: Beyond attack trees: dynamic security modeling with Boolean logic driven markov processes (bdmp). In: Dependable Computing Conference (EDCC), 2010 European, pp. 199–208, April 2010
12. Pudar, S., Manimaran, G., Liu, C.C.: Penet: a practical method and tool for integrated modeling of security attacks and countermeasures. Comput. Secur. **28**(8), 754–771 (2009). http://www.sciencedirect.com/science/article/pii/S0167404809000522
13. Ruddle, A., et al.: Security Requirements for Automotive On-board Networks Based on Dark-side Scenarios. Technical report. Deliverable D2.3, EVITA Project (2009)
14. Schneier, B.: Attack Trees: Modeling Security Threats, December 1999
15. Vigo, R., Nielson, F., Nielson, H.: Automated generation of attack trees. In: 2014 IEEE 27th Computer Security Foundations Symposium (CSF), pp. 337–350, July 2014
16. Zhao, S., Li, X., Xu, G., Zhang, L., Feng, Z.: Attack tree based android malware detection with hybrid analysis. In: IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 380–387, September 2014