http://dx.doi.org/10.14236/ewic/ICS2018.1

CPS Security Assessment using Automatically Generated Attack Trees

Wouter Depamelaere
KU Leuven, imec-DistriNet
Dept of Computer Science, Belgium
wouter.depamelaere@kuleuven.be

Laurens Lemaire
KU Leuven, imec-DistriNet
Dept of Computer Science, Belgium
laurens.lemaire@cs.kuleuven.be

Jan Vossaert
KU Leuven, imec-DistriNet
Dept of Computer Science, Belgium
jan.vossaert@cs.kuleuven.be

Vincent Naessens
KU Leuven, imec-DistriNet
Dept of Computer Science, Belgium
vincent.naessens@cs.kuleuven.be

Over the last decades, cyber-physical systems have evolved from isolated to complex interconnected systems that are remotely accessible. This made them easier to attack, especially since they often contain legacy components with known vulnerabilities. This paper presents a methodology to assess the security of a cyber-physical system. It automatically generates attack trees based on the system architecture. The generated attack trees are processed to provide both technical and non-technical feedback. The assessor can define different attacker models to asses the security of the system with respect to different types of attackers. The methodology is validated by providing tool support and applying it to an example ICS.

Attack Trees, Cyber-Physical Systems, Security Assessment

1. INTRODUCTION

Cyber-physical systems (CPS) are networks of interacting elements with physical input and output, usually containing various geographically distributed field sites. Each field site consists of sensors and actuators, controlled by a programmable logic controller (PLC) or similar device. These remote sites are typically connected to a centralized control network where operators can remotely monitor and control the processes. They fulfill critical functions in the public infrastructure (e.g. electricity distribution, traffic lights) and manufacturing processes. The high potential impact of attacks on these types of systems has made them a popular target for attackers. Examples are the Maroochy Shire sewage spill in Australia, and the Stuxnet worm in Iran. Recently, the Ukrainian grid was compromised which caused hundreds of thousands of people to be without electricity for several hours. Securing these types of systems is not a trivial task, due to their complexity and legacy components with known vulnerabilities.

Threat/attack trees were first proposed in the mid-nineties, and have since become a commonly used tool to assess threats to computer systems. The concept has been further extended by subsequent research, for instance, focusing on incorporating countermeasures or integrating the potential impact of attacks in the tree, tuning them to be used in risk management systems. One of the main disadvantages is that they are, typically, manually constructed making the quality of the analysis significantly dependent on the experience and security background of the assessor. Moreover, attack trees for real-life systems can quickly grow excessively large making them hard to interpret and extract meaningful information, especially for assessors with little security background.

This paper presents a methodology for CPS security assessment. The contribution is threefold. First, attack trees are automatically generated based on the system model and an attacker goal selected by the assessor. Generic templates representing attack patterns are used to iteratively refine the attacker goal. The system model is provided via SysML, an intuitive modeling language for cyber-physical systems. Second, the generated attack trees are automatically assessed to give both technical and non-technical feedback. The assessor can define a custom attacker model to assess the security with respect to different types of attackers. The methodology not only provides an indication of the security of the system against specific types of attackers, it also provides the assessor with a tailored selection of countermeasures that could be implemented to increase the security of the system. Third, adoption of the methodology is facilitated via tool support and it is validated using an example ICS.

Section 2 presents related work. Section 3 gives a general overview of the methodology, after which the attack tree generation and assessment are discussed in more detail in respectively section 4 and 5. Subsequently, the approach is validated in section 6. The paper ends with conclusions.

2. RELATED WORK

One of the most commonly used methods to assess the security of IT systems is attack trees. They were introduced by Schneier (1999), and Mauw and Oostdijk (2005) further refined the foundations of attack trees and presented a formal mathematical definition. The concept of attack trees was extended by Bistarelli et al. (2006) to include defence strategies. They proposed defence trees (DT) in which defences are introduced on the leaf nodes of the attack tree. A slightly different approach, called Attack Defence Trees (Kordy et al. 2011), integrates the defence strategies between the attack nodes, instead of on the leaves. Attack Countermeasure Trees (ACTs) proposed by Roy et al. (2012) are an extension of defence trees, adding branches with detection and mitigation events. Edge et al. (2007) propose to represent the defensive strategies in a separate tree. Other types of attack trees focus on including the consequences of attack scenarios in the attack tree, examples are Attack Response Trees (ARTs)(Zonouz et al. 2009) and Attack Tree with BowTie analysis (ATBT) (Abdo et al. 2018). This facilitates the use of attack trees for risk management. Recently, a new type of attack tree, called Attacker-Manager Game Tree (AMGT), was presented. Traditionally, the root node was the attacker goal and the attack path was determined bottom up. AMGTs represent the rational steps an attacker and a defender would take as if they were playing a game. This model takes into account the cost of each attack step for the attacker and doesn't assume that a countermeasure perfectly inhibits an attack step, as opposed to ACTs and DTs (Arghavani et al. 2018). While attack trees were originally used for IT systems, they can also be applied to cyber-physical systems. This is illustrated by Byres et al. (2004), who use attack trees for risk assessment of SCADA systems. This was followed by other similar work (Ten et al. 2007; Ji et al. 2016). Our work focuses on the automatic generation of attack trees for cyber-physical systems using templates representing common attack patterns. We further focus on automatically providing intuitive feedback to both non-technical and technical users.

Several tools exist that focus on the security analysis of CPS. Some tools such as ADVISE (LeMay et al. 2011), CyberSage (Vu et al. 2014) and CySeMoL (Sommestad et al. 2013) focus on a probability analysis of an attacker reaching the attacker goal. These systems do not provide suggestions to improve the security of the system. ADVISE requires profound security knowledge, since the assessor needs to provide the attack tree as input. The results of the evaluation are largely dependent on the quality of the provided attack tree. The output of CySeMoL is similar to ADVISE, but does not rely on an attack tree provided by the assessor. Instead, generic blocks are defined that allow the assessor to model the CPS system. This reduces the reliance on the security background of the assessor. An important drawback, however, is that CySeMoL uses a fixed attacker model, decreasing the flexibility for the assessor to analyse the security with respect to different types of attackers. In CyberSage, the assessor must provide the workflow (i.e. a series of attack steps to reach the attacker goal). This also requires significant security background, and does not consider alternative attack paths. Other tools focus on compliance with standards. For instance, the Cyber Security Evaluation Tool (CSET) (ICS-CERT 2014) consists of a survey and evaluates the ICS network architecture to assess the compliance of the system with one or more selected standards. The system does not support the modelling of an attacker nor does it provide feedback regarding the security of the system.

Several papers discuss how security feedback can be visualised in security dashboards to users. Kolomeec et al. (2017) define visualization models for information such as attacks, network data and countermeasures. The authors stress the importance of isolating a single security characteristic in metrics. Our methodology defines several security metrics that each give a unique view on the security of the cyber-physical system. McKenna et al. (2016) state that most security analysis tools focus on the data analysis, and neglect tuning the obtained information for the diverse user profiles, such as technicians and managers. The methodology (and tool support) presented in this paper focuses on providing both technical and non-technical feedback. The system does not require a significant security background but can be customized for advanced use cases by expert users.

3. APPROACH

Our approach extends FAST-CPS (Lemaire et al. 2014, 2015), which focuses on analyzing the security impact of known vulnerabilities on the security of

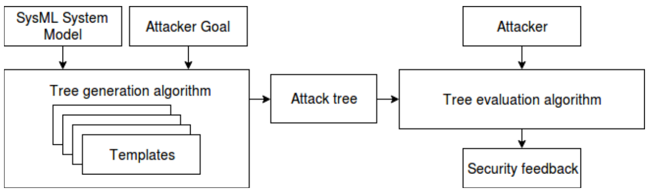


Figure 1: *General overview of the methodology.*

cyber-physical systems. It enables the user to model the system architecture in SysML, a modelling language derived from UML used for model-based systems engineering (Friedenthal et al. 2014) and translates it to a logic-based model. Figure 1 shows a general overview of the methodology. The process is divided in two parts: a *tree generation algorithm* and a *tree evaluation algorithm*. The tree generation algorithm takes as input a FAST-CPS SysML model of the cyber-physical system and an attacker goal. The algorithm generates an attack tree based on the provided input. It uses generic *Templates* that enable iterative refinement of the attacker goal until the attack tree consists of nodes that can no longer be refined. The templates represent common attack patterns and enable generation of the attack tree without human intervention. The generated attack tree is used by the tree evaluation algorithm to give security feedback to the assessor. The security feedback is provided based on the attacker model defined by the assessor. Different types of attackers can be modelled, each with their own capabilities. The algorithm provides both feedback for technical users (e.g. system administrators) and for non-technical users (e.g. plant managers). The tree generation and tree evaluation is presented in more detail in the following two sections.

4. ATTACK TREE GENERATION

This section discusses the automatic generation of attack trees. The generated tree is used in the following section to assess the security of the cyber-physical system against specific attackers. The generated attack tree follows the structure as defined by Mauw and Oostdijk (2005). The nodes of the attack tree represent attacker goals, the root node of the tree represents the global goal of the attacker. Each goal consists of a string representing a general description of the attacker goal and the set of elements of the CPS on which the attacker goal applies (e.g. gain physical access to device PLC1). Other nodes are refinements of this goal, and leaves represent attack goals that can no longer be refined. A parent node can specify either an OR or an AND relation, denoting whether or not it is sufficient for an attacker to complete one child goal, or all child goals need to be completed in order to complete the goal represented by the parent node.

This section first discusses the input required for the attack tree generation. The following subsection discusses how templates are constructed, after which the tree generation algorithm is presented. The final subsection presents a few examples of templates that we defined as part of the methodology and integrated in the tool support (see Section 6).

4.1. Input

The tree generation algorithm requires two types of input: the *system model* and the *attacker goal*.

4.1.1. System Model

The attack trees are generated based on a model of the ICS, which must be provided by the assessor. This paper does not focus on the modelling of the CPS itself but uses an existing modelling framework, namely FAST-CPS (Lemaire et al. 2015). This framework was selected because it allows the assessor to model the system using a graphical formalism based on SysML and automatically links known vulnerabilities extracted from vulnerability databases on the Internet (e.g. the ICS-CERT vulnerability database) to the elements in the system. The framework, further, translates the SysML model of the system to an Imperative Declarative Programming (Wittocx et al. 2008) (IDP) model of the system. IDP[1] is a Knowledge Base System for the FO(·) language. FO(·) is an extension of first-order logic with types, aggregates, inductive definitions, etc. An IDP instance consists of a *Vocabulary* a *Theory*, and *Structure*. In the vocabulary, the FAST-CPS framework defines the non-logical symbols (i.e. types, predicates, functions) that are used to model the system. In the structure, the input model is defined. This is generated based on the SysML model of the ICS and the vulnerability databases. It contains a set of variables E representing the elements in the CPS and instances of the predicates defined in the *Theory* to represent the properties and relations between the different elements. Example types of elements are *HardwareComponent* (i.e. hardware elements in the CPS such as a switch and a PLC), *SoftwareModule* (i.e. a software module such as an operating system or HMI software running on components), *Parameter* (i.e. a CPS process is defined by several invariants expressed in function of thresholds, environment variables and the status of actuators, these are referred to as parameters) and *Network* (i.e. a network that links a set of components). Example predicates specifying the relation between elements are *NetworkLocation(HardwareComponent,Network)* (i.e. indicates that a hardware element has access to a specific network) and *ControlTask(SoftwareModule, Parameter)* (i.e. indicates that the software module is used

[1]More information about the IDP system can be found on `https://dtai.cs.kuleuven.be/software/idp/`.

to control a specific parameter). The theory contains the logic rules expressed that are evaluated by the IDP logic engine. These logic rules can be expressed in a superset of the FO(·) language containing the predicates defined in the vocabulary. The IDP model of the system generated by FAST-CPS is used as input for the tree generation algorithm.

4.1.2. Attacker Goal

The assessor selects an attacker goal that becomes the root of the attack tree. The selected attacker goal is then further refined using the templates, effectively generating the attack tree. If an assessor wants to reason about multiple attacker goals, a separate tree is built for each goal. Two main attacker goals were identified based on the NIST guide to ICS security Stouffer et al. (2015): *ModifyCPSBehavior* and *ObtainData*. The former denotes an attacker attempting to modify the behavior of the CPS process. The latter denotes an attacker attempting to extract any data assets from the CPS. These are fairly generic attacker goals relevant to any CPS system. An assessor can also specify more fine-grained attacker goals by selecting a template and the system elements required as arguments for the template (see Section 4.2). The generic attacker goals match with templates that do not require the assessor to select any parameters.

4.2. Template Model

Each template represents a refinement of a specific attacker goal. A template specifies a unique string representation of the attacker goal and a set of parameters representing the elements in the system model on which this attacker goal should be applied. The values of these parameters are specified by the tree generation algorithm. Each template contains an attack tree representing a refinement of the template's attacker goal, with additional information contained in the nodes. Two types of nodes are defined: final tree nodes F and nodes R that need to be refined further via additional templates.

Final tree nodes contain a string describing the attacker goal represented by the node and a FO(·) expression using the FAST-CPS vocabulary that specifies the (set of) system element(s) on which the attack step applies. The expression defines an equivalence relation specifying the system elements for which the *Apply*(E_1,...,E_n) predicate holds based on the properties and relations between the system elements. This predicate was added to the FAST-CPS vocabulary to be used in templates and the template evaluation (see the following subsection). Nodes that need to be refined further using additional templates contain a reference to the template representing the refinement of the attacker goal. Similar to final tree nodes, they also contain a FO(·) expression that is used to define the system elements that should be provided as parameters to the referenced template.

Figure 2 illustrates how templates can be visually represented by showing the template *ModifyParameter(p)* as an example. Rectangles with dashed lines denote nodes that need to be refined further via templates. Rectangles with full lines denote final nodes of the attack tree. The *#Int* notation binds parameters in the template description or template reference with the parameters in the *Apply* predicate, the integer denotes that order in which they appear in the predicate.

4.3. Tree Generation Algorithm

The tree generation algorithm (see Algorithm 1) uses the attacker goal (AG) and the CPS model (SM), both specified by the assessor, and a predefined set of templates representing generic attack patterns as input to generate the attack tree. The set *goals* represents nodes that need to be refined further using templates. Initially there is one node in the set, which is the attacker goal chosen by the assessor. The algorithm continuously selects nodes that need to be refined further, until the set *goals* is empty. Once a node is selected, the algorithm retrieves the template referenced by the node. The algorithm then evaluates each node in the template. This evaluation starts with the execution of the FO(·) expression associated with the node in the scope of the provided system model and the parameters passed to the template. This results in sets of parameters that satisfy the *Apply* predicate. If no set of parameters that satisfies the predicate is found, the evaluation returns no nodes. Otherwise, the evaluation function returns a node that can be added to the attack tree for each set of parameters that satisfies the predicate. These nodes represent resolved versions of the template nodes in which values are assigned to the parameter declarations in the node. It is possible that one template node results in more than one resolved node after evaluation, since multiple parameter sets can be found that satisfy *Apply*. The *updateTree* method replaces the template node with the resolved versions of the node. If the node represents an attacker goal that needs to be refined further, the resolved nodes are added to the *goals* set. Once the template is resolved, the *goal* node is replaced with the tree generated based on the template referenced by the node. Once *goals* is empty, *tree* is the fully refined attack tree.

4.4. Example Attack Pattern Templates

This subsection shortly discusses a few of the templates we defined as part of the methodology and which are also integrated in the tool support

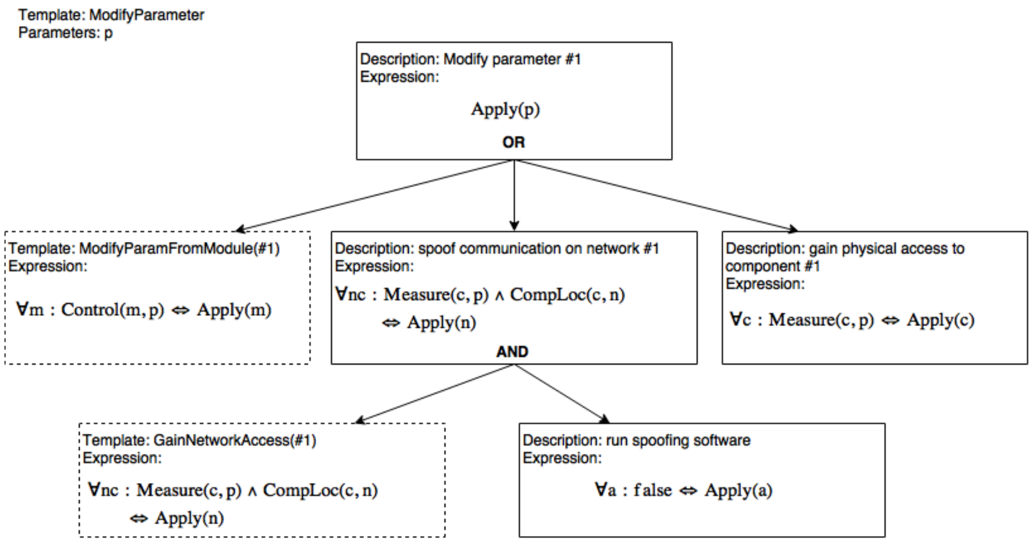

Template: ModifyParameter
Parameters: p
Description: Modify parameter #1
Expression:
Apply(p)
OR
Template: ModifyParamFromModule(#1)
Expression:
∀m : Control(m, p) ⇔ Apply(m)
Description: spoof communication on network #1
Expression:
∀nc : Measure(c, p) ∧ CompLoc(c, n) ⇔ Apply(n)
AND
Description: gain physical access to component #1
Expression:
∀c : Measure(c, p) ⇔ Apply(c)
Template: GainNetworkAccess(#1)
Expression:
∀nc : Measure(c, p) ∧ CompLoc(c, n) ⇔ Apply(n)
Description: run spoofing software
Expression:
∀a : false ⇔ Apply(a)

Figure 2: *Template ModifyParameter(parameter).*

Algorithm 1 GenerateTree(AG,SM,Templates)

```
Tree $tree$ = new Tree();
$goals = \{AG\}$
while $goals \neq \emptyset$ do
    $goal$ = $goals$.pop();
    Template $t$ = getTemplate(goal);
    for all TemplateNode $tn \in t$ do
        Node[] $n$ = $tn$.eval($SM, goal.params$);
        $t$.updateTree($tn, n$);
        if $tn \in R$ then
            $goals$.push($n$);
        end if
    end for
    $tree$.replace($goal$, $t$.getTree());
end while
```

discussed in Section 6. A full overview of the 12 templates we currently defined and integrated in the tool can be found in Depamelaere (2018). The system is sufficiently flexible to allow third-parties to extend the system with additional templates, increasing the scope of the analysis.

- *ModifyCPSBehavior*: this template infers the different parameters monitored and controlled by the CPS and, for each parameter, generates a node referencing the *ModifyParameter(parameter)* template.

- *ObtainData*: this template infers the different data assets defined in the system model and further refines the attacker goal by generating a node referencing the *Obtain(data)* template for each data asset.

- *ModifyParameter(parameter)*: this template represents an attacker changing the behaviour of the CPS process by modifying a process parameter. The template is shown in Figure 2. This template takes into account which software modules are able to control the specified parameter, which networks they are located in, and which hardware components are sensors that measure the specified parameter. How many children the root node has depends on the system architecture. For every software module that can modify the parameter, a child node is added. These nodes are further matched with another template which will extend the tree. Similarly, for every sensor that measures the parameter a leaf node is added, representing an attacker modifying the parameter by obtain physical access to the sensor/actuator related to the specified parameter. The attacker could also attempt to spoof communication from a component to the sensor/actuator. For this route to succeed, the attacker must gain access to a network used by the component to interact with the sensor/actuator (refined in the *GainNetworkAccess(n)* template) and spoof the communication.

5. ATTACK TREE ASSESSMENT

Attack trees for real-life systems quickly grow in size to a point at which they are very hard to interpret by human assessors. Moreover, determining what useful information can be extracted from attack trees is not trivial for assessors with a limited security background. Hence, this section focuses on the automated analysis of attack trees to provide intuitive security feedback to both system administrators (technical feedback) and managers (non-technical feedback). This section first discusses the attacker model used during the attack tree analysis. Subsequently, the strategy for assigning a difficulty level to each attack step (i.e. attacker goal) is described. The section ends with a discussion of the security feedback presented to the assessor.

5.1. Attacker Model

The tree assessment requires the input of the attack tree (automatically generated as described in the previous section) and an attacker model. The attacker model $\mathcal{A}$ = *(C, M, A)* constitutes a set of credentials *C* available to the attacker, a set of capabilities *M* (each mapped to an integer value ranging from 1 (no expertise) to 4 (expert)), and a set of components in the system *A* which the attacker has physical access to. The following attacker capabilities are considered:

- *Stealing Credentials*: Stealing credentials from other users in the system by performing a phishing attack, social engineering, etc.
- *Identity Spoofing*: Pretending to be another entity whilst sending commands over a channel.
- *Exploiting Vulnerabilities*: Exploiting known vulnerabilities in software/hardware components.
- *Discovering Vulnerabilities*: Discovering new vulnerabilities in software/hardware components.

5.2. Node Difficulty Assessment

The node difficulty ranges from 1 (trivial) to 4 (unlikely) and indicates how hard it is for the modelled attacker to reach the attack goal represented by that node. The difficulty assignment consists of two steps. First a difficulty is assigned to each node based on the attacker capabilities. After that, the impact of countermeasures is reflected in the node difficulty.

5.2.1. Attacker-Based Difficulty

The first stage of the difficulty assignment is the calculation of the difficulty for each leaf node. Two types of leaf nodes can occur. The first type is related to physical access to a component or the possession of a credential. If the attacker has the required credential or access, the leaf is automatically labelled as *trivial* (i.e. difficulty 1), and *unlikely* (i.e. difficulty 4) if otherwise. The other type represents an attack goal that requires a particular skill (e.g. exploit vulnerabilities). These steps become easier the more proficient the attacker is. If an attacker is an expert in the skill required for a particular attack step, then its difficulty is set to 1. Decreases in skill result in proportional increases in difficulty. For nodes with children in an OR-relation, the difficulty of the node with the lowest difficulty is assigned, since only one of the child goals must be completed to complete the parent goal. For nodes with children in an AND-relation, all child goals need to be accomplished to reach the parent node. Hence, the highest difficulty of the child goals is selected.

5.2.2. Attack Countermeasure Tree

System administrators often take measures to harden the security of their system. Not all of these measures are related to system architecture and, hence, contained in the SysML model. Examples are password policies, software update policies and intrusion detection systems. A set of 26 of these types of countermeasures was selected based on documents from ENISA (2016) and ICS-CERT (2011). These countermeasures can be applied to both leaf and intermediary nodes, following the Attack Countermeasure Tree specification (Roy et al. 2012). Not all countermeasures apply to all types of nodes. The activation of a countermeasure on a node results in a difficulty increase of that node with at most 3. Apart from a difficulty increase, each countermeasure also has an attribute reflecting the complexity of implementation (as suggested by ENISA). This property has 3 levels, ranging from *low* to *high*. This is used in the following subsection where intelligent suggestions are given on which countermeasures could be applied to further increase the security of the system. A detailed overview of the available countermeasures, on which types of nodes they can be applied, the implementation complexity and the associated difficulty increase can be found in Depamelaere (2018). After a countermeasure is applied to a node, the difficulty update is propagated up the tree.

5.3. Security Feedback

This section discusses three types of security feedback based on the attack tree and attacker model, targeting both non-technical and technical users.

5.3.1. Easiest Attack Path

Determining the most probable attack path for a given attacker isn't straightforward for assessors in large attack trees. Hence, the system automatically marks the attack path with the lowest difficulty. This is calculated using the recursive algorithm listed in Algorithm 2.

5.3.2. Security Metrics

This paragraph defines a set of security metrics that give an indication of the security status of the system to non-technical users. The security metrics are especially relevant for assessors that want to compare multiple alternative system architectures or determine the impact of countermeasures (see following subsection).

- *Global difficulty* matches the *difficulty* parameter of the root node. It represents the difficulty for the modelled attacker to reach the global attacker goal. A higher global difficulty generally means a more secure system.

Algorithm 2 MarkEasiestPath(n)

```
$n.easy \leftarrow true$
**if** $n.children \neq \emptyset$ **then**
  **if** $n.operation = AND$ **then**
    **for all** $c \in n.children$ **do**
      MarkEasiestPath($c$)
    **end for**
  **else if** $n.operation = OR$ **then**
    **for** $c \in n.children|\forall c' \in n.children\backslash\{c\} : c'.difficulty > c.difficulty$ **do**
      MarkEasiestPath($c$)
    **end for**
  **end if**
**end if**
```

- The *difficulty distribution* represents the number of nodes in the tree marked with each level of difficulty. For setups with equal global difficulty, the assessor could prefer a setup where the attacker is more likely to quickly reach attacker goals with a higher difficulty. The distribution not only considers leaf nodes because countermeasures can affect the difficulty of intermediary nodes.

- *Security score* is a double between zero and three representing a weighted average of the difficulty of the attack steps. The easiest nodes don't contribute to the score and have weight zero.

$$score = \frac{\sum_{d=1}^{4} (d-1)\cdot \mid \{n|\forall n \in nodes, n.diff = d\} \mid}{\mid nodes \mid}$$

5.3.3. Smart Countermeasure Suggestion
Selecting the most effective countermeasure to impede an attacker isn't trivial. This section discusses how the system proposes countermeasures tailored to mitigate the attacks on the modelled system. The algorithm used to gather the set of countermeasure suggestions is shown in Algorithm 3.

First, the nodes along the easiest attack path are selected. From this subset, the nodes that have the same difficulty level as their parent are selected as they represent the weakest nodes. For each of these nodes, the best available countermeasure is selected (if there are any). The parameter used to determine the *best* countermeasure is selected by the assessor from two options: *diff-increase* and *complexity*. The former favours countermeasures with the highest increase in difficulty for the node on which it is applied, the latter favours countermeasures that are easier to implement. The resulting list is then sorted by impact on *security score*, so the countermeasures with the most impact on the difficulty distribution are suggested first. For countermeasures that result in an equal security score, the *best* criterion is used as a tiebreaker.

Algorithm 3 GatherBestCountermeasures($emph$)

```
**Require:** $emph \in \{$*diff-increase*, *complexity*$\}$
**Output:** $BestCMList$
  $\{$*Find best CM's on attack path*$\}$
  **for all** $n \in$ attack-path **do**
    **if** $n.CM \neq \emptyset$ and $n.diff \leq n.parent.diff$ **then**
      $A \leftarrow$sort $\{cm|cm.active = F, \forall\ cm \in n.CM\}$ by $emph$
      $BestCMList \leftarrow BestCMList \cup A.first$
    **end if**
  **end for**
  $\{$*Simulate and sort by score and emphasis*$\}$
  $D \leftarrow deepcopy($*attack tree*$)$
  **for all** $cm \in BestCMList$ **do**
    enable $cm$ in $D$
    save attacker goal difficulty and score
    disable $cm$ in D
  **end for**
  sort $BestCMList$ by score, then by $emph$
  **return** $BestCMList$
```

6. VALIDATION

This section validates the methodology via a case study and a discussion of the tool support (the source code can be found in Depamelaere (2018)) provided to facilitate adoption of the methodology.

6.1. Illustrative Example

The example ICS is a pumping station with a small network (see Figure 3) for the sake of simplicity. The main activity of this infrastructure is to keep the water level of a river between certain boundaries using pumps. This particular system is heterogeneous, as most cyber-physical systems. First of all, there is a control network with a water level sensor, pump actuator, HMI panel and a PLC controller. There is also a supervisory network containing a data historian and an engineering workstation. The connection between both is made using switches and routers. The input file used for the example can be found in Depamelaere (2018).

The selected attacker goal for the use case is the manipulation of the pump. For the evaluation, three attacker profiles were chosen (see Table 1). The first attacker is an experienced hacker external to the system. This implies that he is proficient in the defined capabilities, but has no physical access to components. The second attacker is a member of the cleaning staff and, hence, has access to several components but scores low on the capabilities. Neither attacker possess any ICS credentials.

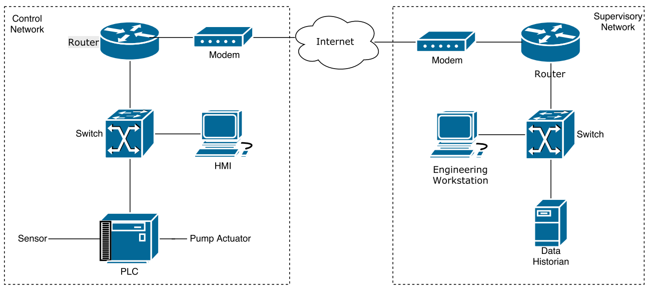


Figure 3: *Pumping station network architecture.*

The third attacker profile reflects the collaboration between the hacker and the cleaning staff. The hacker, for instance, can bribe an employee to obtain physical access to system components.

The results of the case study are shown in Table 2. The difficulty distribution is listed in the second column. The generated attack tree consists of 117 nodes. In the first part of the table, the security status of the system is listed for each attacker profile. The CPS scores the highest possible global difficulty for each individual attacker profile. However, if both profiles are combined, the attacker goal difficulty drops to 2, and the security score to 1.7. The second part of the table shows the impact of applying the first suggested countermeasures (CM) incrementally for each attacker profile. For the hacker, the first suggested countermeasure is to install an anti-virus on the engineering workstation, protecting the password needed to change the parameter by impersonating an employee. This increases the score with 0.1. The threat originating from the cleaning staff member is mainly caused by the physical access he possesses. Therefore, the suggested countermeasures are to prevent unnecessary access to these components, which greatly improves the security. These are also the most important countermeasures for the combined attacker profile. Once these are secured, the countermeasure related to the passwords is proposed. Additional countermeasures (e.g.input validation) can further increase the security, but are not mentioned in the table for reasons of brevity. The default sorting of countermeasures is by security score. In the last row of the table, the result is shown if the assessor focusses on the goal difficulty. The suggested countermeasure applies to the root node, which directly protects the attacker goal but leaves many sub nodes unprotected. The attackers still have access to the vital components, and could still cause serious damage.

6.2. Discussion

This subsection discusses the tool support provided to the assessor to facilitate adoption of the presented methodology. The assessor first models the CPS in SysML and converts it to an IDP model using the FAST-CPS framework. The tool reads the output file generated by FAST-CPS, containing the CPS model. The assessor then selects an attacker goal based on the data in the CPS model. The attacker can be modelled by selecting the credentials and physical access he possesses and defining his capabilities. Multiple attackers can be modelled, labelled and stored, each with different profiles (e.g. malicious employee, script kiddie and dedicated hacker). The profiles can be activated separately or simultaneously to simulate collaborations. Modifications to the input model trigger a regeneration of the attack tree, followed by an update of the security feedback.

The tool provides a comprehensible dashboard (see Figure 4) that provides an at-a-glance overview of the security metrics and the suggested countermeasures. A first security metric is the difficulty distribution, presented as a tile with a donut chart. The second tile has a gauge that indicates the security score. The global difficulty is the most obvious indicator and is also displayed by a gauge tile. A fourth tile displays the capabilities of the (compound) attacker. The lower part of the

Table 1: *The attacker models used in case study.*

Profile	*'hacker'*	*'Cleaning staff'*
Capabilities		
Obtaining credential	3	1
Discover zero-day vuln.	2	1
Exploit zero-day vuln.	3	1
Spoof protocol	2	1
Physical Access	-	*PLC, Router*
Credentials	-	-

Table 2: *The security feedback of the case study.*

Situation	**Difficulty distribution**	**Sec. score**	**Total diff.**
Without CMs	*[4]-[3]-[2]-[1]*		
Only 'hacker'	77-22-18-0	2.5	4
Only 'cleaning staff'	89-0-0-28	2.3	4
Both	38-30-21-28	**1.7**	**2**
With CMs			
HACKER			
Install A-V (Workstation)	83-27-7-0	2.6	4
CLEANING STAFF			
1. Phys. Isol. 'Router'	95-0-0-22	2.4	4
2. Phys. Isol. 'PLC'	117-0-0-0	3.0	4
BOTH			
Focus: Security score			
1. Phys. Isol. 'Router'	44-30-21-22	1.8	2
2. Phys. Isol. 'PLC'	77-22-18-0	2.5	4
3. Install A-V (Workstation)	83-27-7-0	**2.6**	**4**
Focus: Goal difficulty			
Install IDS/IPS	39-30-20-28	**1.7**	**4**

window is dedicated to the list of countermeasures suggested to the assessor. Each countermeasure tile displays the security score of the system if applied. The contour of the panels indicate the goal difficulty if applied. The users can select which countermeasures should be applied to the system, and immediately receives feedback on the impact via an update of the security metrics. Additional information is shown in separate tabs, such as an overview of the attackers. The modelled attackers can be enabled or disabled, resulting in an update of the security metrics. A visual representation of the attack tree in which the easiest path is highlighted is shown in a separate tab.

The algorithm for drawing the attack tree uses a post-order traversal. Every leaf node is drawn to the right of the previous leaf node (regardless of depth) and once all children of a node are drawn, the parent is centred above them. To account for the quick increase in size, the tool supports collapsing of subtrees by the user. The generated trees and modelled attackers can be saved in JSON format, to be imported later. Each template is implemented in a separate Java class, with an execute method in which the template's parameters are passed and which returns the list of resolved nodes.

7. CONCLUSIONS

This paper presented a methodology for the security evaluation of cyber-physical system. The methodology automatically generates attack trees using generic templates representing typical attack patterns. This enables the assessor to easily compare different versions of a system model or simulate the impact of updates on the model. The assessor effort is shifted from updating the attack trees itself to modifying the system model in SysML, which is more intuitive and less error-prone. The system is also extensible, allowing third-parties with a security background to define additional templates or modify existing templates to tune the system to their specific needs and refine the security feedback. The flexibility to add new templates is, however, restricted by the logical model of CPS systems used by the FAST-CPS framework. If new concepts are required to express desired attack patterns, the FAST-CPS model needs to be extended. The generated attack tree is automatically assessed to provide intuitive security feedback to the assessor. Tool support is provided to facilitate adoption of the methodology and it was applied to a case study. The tool contains a security dashboard providing an at-a-glance overview of the security of the system, suitable for managers and other stakeholders with a less technical background. Further, tailored countermeasures are suggested that can increase the security of the system.

REFERENCES

Abdo, H., M. Kaouk, J.-M. Flaus, and F. Masse (2018). A safety/security risk analysis approach of industrial control systems: A cyber bowtie combining new version of attack tree with bowtie analysis. *Computers and Security 72*, 175 – 195.

Arghavani, A., M. Arghavani, M. Ahmadi, and P. Crane (2018). Attacker-manager game tree (amgt): A new framework for visualizing and analysing the interactions between attacker and network security manager. *Computer Networks 133*, 42 – 58.

Bistarelli, S., F. Fioravanti, and P. Peretti (2006, April). Defense trees for economic evaluation of security investments. In *First International Conference on Availability, Reliability and Security (ARES'06)*.

Byres, E. J., M. Franz, and D. Miller (2004). The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of the international infrastructure survivability workshop*.

Depamelaere, W. (2018). Javafx tool for attack trees in cps, `https://github.com/WouterDep/attacktrees`.

Edge, K., R. Raines, M. Grimaila, and R. Baldwin (2007). The use of attack and protection trees to analyze security for an online banking system. In *Proceedings of the 40th Hawaii International Conference on System Sciences*.

ENISA (2016, December). *Communication network dependencies for ICS/SCADA Systems*.

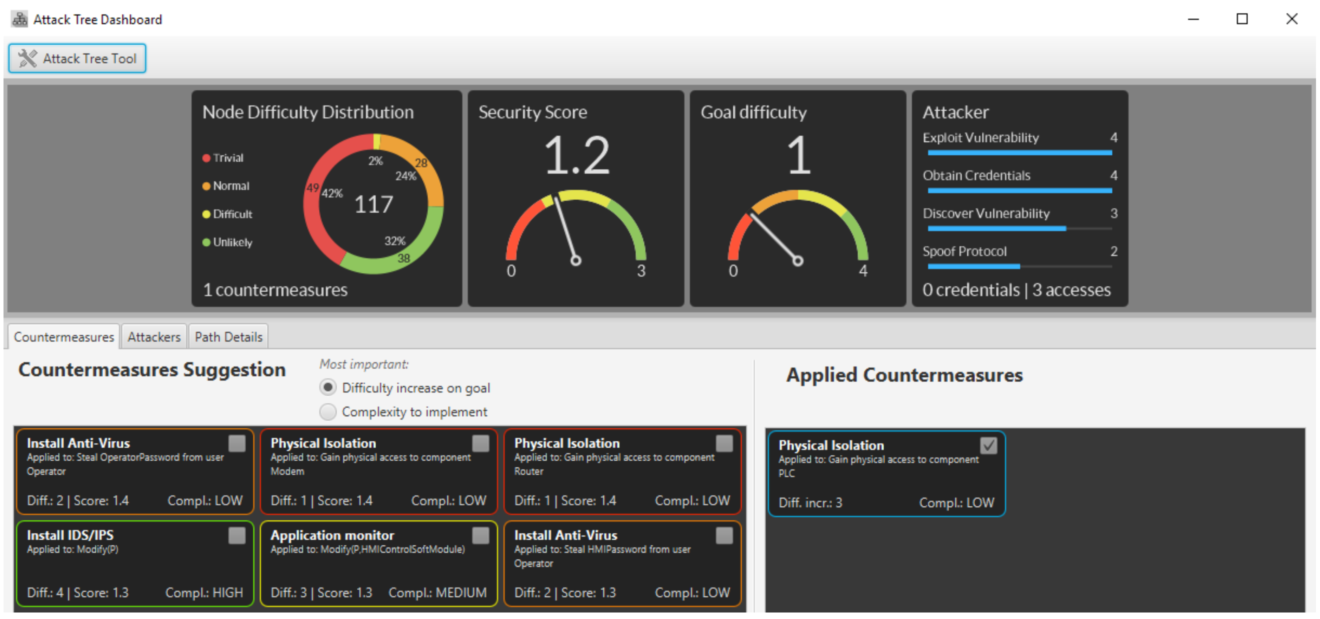
Friedenthal, S., A. Moore, and R. Steiner (2014). *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann.

ICS-CERT (2011, May). *Common Cybersecurity Vulnerabilities in Industrial Control Systems*.

ICS-CERT (2014). CSET: Cyber security evaluation tool, `http://ics-cert.us-cert.gov/Assessments`.

Ji, X., H. Yu, G. Fan, and W. Fu (2016). Attack-defense trees based cyber security analysis for cpss. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on*, pp. 693–698. IEEE.

Kolomeec, M., G. Gonzalez-Granadillo, E. Doynikova, A. Chechulin, I. Kotenko, and H. Debar (2017). Choosing models for security metrics visualization. In *Computer Network Security*, pp. 75–87. Springer.



1 countermeasures

Security Score: 1.2

Goal difficulty: 1

Attacker:
Exploit Vulnerability 4
Obtain Credentials 4
Discover Vulnerability 3
Spoof Protocol 2
0 credentials | 3 accesses

Countermeasures Suggestion

Most important:
[✓ Difficulty increase on goal]
Complexity to implement

Install Anti-Virus
Applied to: Steal OperatorPassword from user Operator
Diff.: 2 | Score: 1.4 Compl.: LOW

Physical Isolation
Applied to: Gain physical access to component Modem
Diff.: 1 | Score: 1.4 Compl.: LOW

Physical Isolation
Applied to: Gain physical access to component Router
Diff.: 1 | Score: 1.4 Compl.: LOW

Install IDS/IPS
Applied to: Modify(P)
Diff.: 4 | Score: 1.3 Compl.: HIGH

Application monitor
Applied to: Modify(P,HMIControlSoftModule)
Diff.: 3 | Score: 1.3 Compl.: MEDIUM

Install Anti-Virus
Applied to: Steal HMIPassword from user Operator
Diff.: 2 | Score: 1.3 Compl.: LOW

Applied Countermeasures

Physical Isolation ✓
Applied to: Gain physical access to component PLC
Diff. incr.: 3 Compl.: LOW

Figure 4: *Dashboard with countermeasure view.*

Kordy, B., S. Mauw, S. Radomirovi, and P. Schweitzer (2011). Foundations of attack-defense trees. *Formal Aspects of Security and Trust*, 80–95.

Lemaire, L., J. Lapon, B. De Decker, and V. Naessens (2014). A SysML extension for security analysis of industrial control systems. In *Proceedings of the 2Nd International Symposium on ICS & SCADA Cyber Security Research 2014*, ICS-CSR 2014, pp. 1–9. BCS.

Lemaire, L., J. Vossaert, J. Jansen, and V. Naessens (2015). Extracting vulnerabilities in industrial control systems using a knowledge-based system. In *Proceedings of the 3rd International Symposium for ICS & SCADA Cyber Security Research*, ICS-CSR '15, pp. 1–10. BCS.

LeMay, E., M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehrcke (2011). Model-based security metrics using adversary view security evaluation (advise). In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pp. 191–200. IEEE.

Mauw, S. and M. Oostdijk (2005). Foundations of attack trees. In *International Conference on Information Security and Cryptology*, pp. 186–198. Springer.

McKenna, S., D. Staheli, C. Fulcher, and M. Meyer (2016, 6). Bubblenet: A cyber security dashboard for visualizing patterns. *Computer Graphics Forum 35*(3), 281–290.

Roy, A., D. S. Kim, and K. S. Trivedi (2012, August). Attack countermeasure trees (act): Towards unifying the constructs of attack and defense trees. *Sec. and Commun. Netw. 5*(8), 929–943.

Schneier, B. (1999). Attack trees. *Dr. Dobbs journal 24*(12), 21–29.

Sommestad, T., M. Ekstedt, and H. Holm (2013). The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures. *Systems Journal, IEEE 7*, 363–373.

Stouffer, K., S. Lightman, V. Pillitteri, M. Abrams, and A. Hahn (2015). Guide to industrial control systems (ics) security.

Ten, C.-W., C.-C. Liu, and M. Govindarasu (2007). Vulnerability assessment of cybersecurity for scada systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pp. 1–8. IEEE.

Vu, A. H., N. O. Tippenhauer, B. Chen, D. M. Nicol, and Z. Kalbarczyk (2014). Cybersage: a tool for automatic security assessment of cyber-physical systems. In *International Conference on Quantitative Evaluation of Systems*, pp. 384–387. Springer.

Wittocx, J., M. Mariën, and M. Denecker (2008). The idp system: a model expansion system for an extension of classical logic. In *Proceedings of the 2nd Workshop on Logic and Search*, pp. 153–165.

Zonouz, S. A., H. Khurana, W. H. Sanders, and T. M. Yardley (2009, June). A game-theoretic intrusion response and recovery engine. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, pp. 439–448.