Model-Based Risk Analysis Approach for Network Vulnerability and Security of the Critical Railway Infrastructure

Himanshu Neema[1(✉)], Leqiang Wang[1], Xenofon Koutsoukos[1], CheeYee Tang[2], and Keith Stouffer[2]

[1] Vanderbilt University, Nashville, TN 37212, USA
himanshu.neema@vanderbilt.edu
[2] National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
cheeyee.tang@nist.gov

Abstract. This study focuses on threat modeling, vulnerability analysis, and risk management within the critical railway transportation infrastructure. The Railway Transportation System is a highly complex, national critical infrastructure and its cybersecurity evaluation is crucial, but is still an extremely hard problem. In this paper, a novel threat modeling and risk management approach using a domain-specific modeling environment is presented. Two risk analysis techniques based on attack trees are developed to systematically model the potential risks in a cyber-physical system and provide quantitative analysis of the vulnerabilities. The automated risk assessment tool can prioritize component level vulnerabilities for potential mitigation actions. A scenario language and associated tools in the framework allow modeling and evaluation of cyber-games using a library of system exploits and mitigation actions. Cyber-games enable assessment of system-level risks and development of comprehensive risk management plans. Another key capability is the handling of dynamic network connections with variable vulnerability propagation in railway communication networks where locomotives and its devices are mobile. These capabilities are demonstrated with a case study in the railway transportation domain.

Keywords: Risk analysis · Threat modeling · Metamodeling · Vulnerability analysis · Cyber-gaming · Security · Cyber-physical system

1 Introduction

Designing and maintaining safety-critical infrastructures is a challenging task that requires minimizing risks of cyber-attacks and building resilience into their design to keep them operational despite cyber-attacks. For a reliable critical infrastructure, its security assessment and the configurations and arrangement of components must be carefully considered. As security mechanisms do impact

D. Percia David et al. (Eds.): CRITIS 2021, LNCS 13139, pp. 79–98, 2021.
https://doi.org/10.1007/978-3-030-93200-8_5

the system's performance [1], we must evaluate if these mechanisms are necessary and sufficient for the system's cybersecurity [2].

Vulnerability assessment and risk management of critical infrastructure is crucial in the modern world. Ongoing increases in network connectivity, distributed computing and control, and variability of network topology has dramatically increased the attack surface and made vulnerability evaluation a highly complex task, Traditional security analysis by domain experts is largely manual and relies on the judgment of professionals to qualitatively assess system vulnerability. There are several drawbacks of this approach. First, the manual assessment and reliance on personal experience, makes risk models subjective and often inconsistent among organizations. Since an attacker only needs one opportunity to succeed, this inconsistency could lead to a significant problem. Secondly, the manual approach is not scalable as the system grows larger. Considering a system as a graph, with components as its vertices and network interactions among components as its edges, the worst-case complexity of the number of edges is equal to the square of the number of vertices. For risk analysis of a system, both the relations between components and properties of individual components must be analyzed. Manually addressing this complexity is highly challenging, time-consuming, and error-prone.

In this paper, we apply Model-Integrated Computing (MIC) techniques [3] with software tools for quantitative risk analysis of Railway Transportation Systems (RTSs). Traditional Cyber-Physical Systems (CPS) are systems that involve tightly-coupled control, computation, and communication components where the CPS' functionality emerges from the interaction of components. Digital connectivity among CPS' components and their interactions makes their vulnerability assessment difficult. This is even harder in RTSs as these are geographically distributed with continuously changing network topology due to the movement of locomotives and their on-board sensors and devices. We describe our work on designing a web-based Risk Analysis Framework (RAF) for this purpose. RAF is developed using WebGME [4] – a web-based platform that allows not only metamodeling, but also developing custom visualizers and plugins. The framework develops two core components: the RAF metamodel and the risk assessment and visualization tools. The RAF metamodel allows modeling the system architecture with different system components and their network topology, network interconnections among components, the risks and vulnerabilities of CPS components, and cyber-games for dynamic risk management. The RAF analysis plugins read and calculate vulnerability scores and save the internal property data in the system model itself. The RAF visualization tools (called *visualizers*) generate and display (using a novel layout algorithm) risk propagation trees and risk values in WebGME. Both the analysis plugins and visualizers are developed in WebGME and implemented in JavaScript.

The organization of this paper is as follows. Section 2 surveys the related work in the area of cyber-physical system and threat modeling and risk analysis. Section 3 presents the system architecture with a detailed description of key technical issues in the implementation. An example from the railway domain is

presented in Sect. 4. Finally, Sect. 5 concludes the paper and provides directions for future research.

2 Related Work

Threat modeling and vulnerability analysis is an established field with many works. A real-world quantitative vulnerability assessment of critical infrastructures in Norway appeared in [5]. It uses real-world tools for scanning internet connected systems and assesses their vulnerabilities. Our work is focused on *model-based risk analysis* for effective evaluation of *risk management plans*.

Standards and background knowledge for threat modeling in the railway system domain can be found in [6]. It introduces a state-of-art railway framework based on European Railway Traffic Management System (ERTMS) and elaborates the general logic when designing railway threat modeling systems including the modeling of components and processes. The basic concepts of attack trees and risk propagation techniques could be found in [7].

A metamodeling approach [8] for risk analysis in a railway temperature monitoring system model by modeling the railway system and corresponding properties aligns well with our work. This work only covers modeling the *system architecture*, while our work includes the *risk analysis algorithms* as well as *risk management planning*.

A method was presented in [9] for quantifying system-level cybersecurity risk by analyzing the risks at individual system components as well as the information and control flows among them. In our approach, we consider realistic network simulation and network topology that enables a fine-grained evaluation. In addition, our ongoing work (see Sect. 5) is on interfacing model-based risk evaluation with integrated simulation based impact assessments, which requires simulating the network and cyber-attacks.

A consideration of both cyber and physical attack paths appeared in [10]. Even though the use-cases we modeled involved only cyber-attacks, our framework can be directly applied to model physical-attacks.

The idea of chaining vulnerabilities to evaluate impact of exploitation of multiple vulnerabilities in attack paths and prioritizing attach paths was discussed in [11]. However, our approach for dynamic risk management to evaluate multiple attack paths along with mitigation actions is much broader and powerful.

An approach to model attack paths using a weighted colored petri net and modeling threat propagation using incomplete information Bayesian games appeared in [12]. In contrast, our approach uses models to intuitively specify the system architecture and network topology that mirrors what is found in real-world applications, and provides automation tools for vulnerability propagation as well as risk management using attacker-defender games.

The basic concepts of modeling threats using the attack-centric, asset-centric, and software-centric approaches were introduced in [13], and later used for designing a risk analysis method in [14].

Our paper combines theoretical problem modeling with a *real-world scenario* in the railway traffic domain and highlights the core technical issues and presents our solutions for designing a *realistic risk analysis framework*. Importantly, none of the works cited above are able to deal with *dynamic connections* that arise due to changing network topology of systems.

3 System Architecture

3.1 Modeling Approach

A *metamodel* is a model of the model, i.e., a simplified model of an actual model of a circuit, system, or software like entity [3,15]. A metamodel can be a mathematical relation or algorithm representing input and output relations. A model is an abstraction of a phenomenon in the real world; a metamodel is yet another abstraction, highlighting properties of the model itself. A model conforms to its metamodel in the way that a computer program conforms to the grammar of the programming language in which it is written. Various types of metamodels include polynomial equations, neural network, and Kriging. *Metamodeling* is the construction of a collection of *concepts* (things, terms, etc.) within a certain domain and describing how these concepts are related. Metamodeling typically involves studying the output and input relationships, the organization and association of different concepts in the domain, and then designing the right metamodel tools that capture their run-time behavior.

3.2 Modeling Environment

The RAF's metamodel is developed using WebGME [4], which is a web-based, collaborative meta-modeling environment with a centralized version-controlled model storage. WebGME is a client-server based application, where both the client (browser) and server-side (NodeJS) use JavaScript. The clients carry a significant amount of the workload and the role of the server is mainly to store and retrieve the raw model data and propagate events between collaborating clients. A simplified and partial view of RAF's metamodel is shown in Fig. 1.

3.3 Modeling Railway Infrastructure and Communications

Device Components. Devices are basic elements in a system model. A device can contain other children devices, which represents hierarchical decomposition of the system. In metamodel perspective, a device can be either high-level device or base-level device. The base-level devices are basic hardware components contained in the upper-level abstract device component. The design of device components metamodel not only consists of the metamodels for all devices (e.g., sensors and repeaters), but also includes the connections between those components, which are typically network connections (e.g., wired or wireless) only between base devices. Due to the communication property and nature of WebGME's connection type component, the connections are directional from one component to

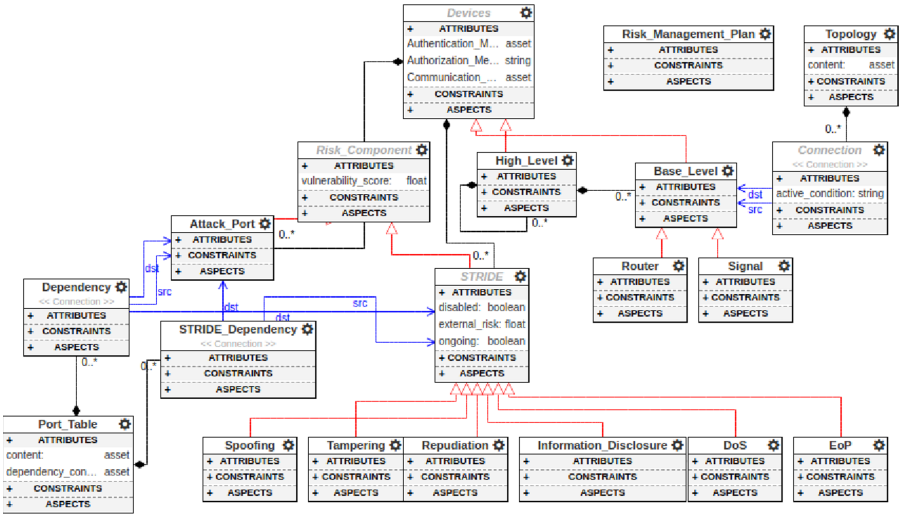

Fig. 1. RAF metamodel (simplified and partial view)

the other; even when these connections in reality are mostly bidirectional. The topology component in RAF allows connecting base-level devices.

For the devices, there is an abstract model called *Device* with a set of inherited nodes representing specific devices. For the railway networks, the device models currently include *Railway Signal*, *Sensor*, *Router*, *Repeater*, *Central Station*, and *Gateway*, and the specific network connections modeled are *WIFI*, *wireless*, and *IP Network*. Three properties of the *Device* meta node are inherited by all devices, viz. *authentication method*, *authorization mechanism*, and *communication protocol*. Authentication method and authorization mechanism both affect the risk propagated from other devices through their connections and the risk spawned within the device itself. Communication protocol is one of the key factors affecting how risks propagate among devices.

Risk Components. The risk components are abstract elements representing a risk that can occur or some intermediate event, or vulnerable points that can be exploited by malicious actors. The external risk analysis is formally modeled using a system attack graph (SAG). We provide risk dependencies and attack ports for it. Risk dependency connects different device components under the crosscut of a graph table. Crosscuts in WebGME allow viewing and connecting metamodel elements spread across containment hierarchy and different modeling pages. The *STRIDE_Dependency* is a connection component type modeled as a child of the first-class object (FCO) in WebGME and is contained in the graph table. *STRIDE_Dependency* has *STRIDE risk* [16] and *attack port* as its source and destination respectively. To model risk to risk dependencies in the dependency graph, we use a separate type of connection called *Dependency* for

STRIDE risk to attack port dependency, and this has both attack port and STRIDE risk as destination nodes, but only attack port as source nodes.

As described before, attack port is an abstraction of some potentially risky behavior that can indirectly cause risk. It can be considered as a specific potential attack surface. Attack port meta node is a direct child of the abstract type risk component, which shares the common property called *vulnerability score*. The attack ports are contained in a device component like the other risk components.

Several other meta nodes are modeled for Component Attack Trees (CAT). The *STRIDE* meta (not shown for brevity) demonstrates how meta nodes relate to risks, specifically how the internal risks, which are analyzed via CATs, are organized. In CATs, the children's risk can be combined using *AND* and *OR* relations. With AND relation, all the children risks must happen together in order to trigger the parent event.

A significant difference in a CAT from a SAG is how vulnerability scores are related. In a SAG node, if one of its children nodes' risk event occurs, its risk event is also considered to have occurred. In other words, all children's risk values are combined with an OR relationship. However, it can also be a combination of AND/OR relations which can be formalized by a logical expression. In an AND/OR logical expression, the AND operator has higher priority than the OR operator. The expression is computed using a tree logic, with AND and OR operands such that each AND operation is treated as an operation inside a bracket, which sits deeper in the computation process in the tree. The flat layer of the computation tree is always an expression combined with OR relations unless there is only one expression. Therefore, we treat a single intermediate node as a special case of the OR expression, and, by default, all children under a node are combined with OR. If there is an AND expression embedded in the whole expression, we model this by putting an *All_Combo* node on top of them, and putting those nodes under it by combining them using AND relations.

Graph Data Component. In order to facilitate easier risk analysis by users, we needed to build automation tools for both the user interaction and data processing. The risk analysis is based on CAT and SAG, both of which use tree-based data structures and their processing time grows exponentially with the system size. Visualizer should be light-weight components with low computational overhead. Therefore, we used a temporary, read-only data structure (called *Graph Data*) to store the system information, and restricted visualizer to read data lazily only before rendering the results. All devices must contain one and only one graph data component. However, as WebGME does not generate contained components while creating a parent component, we must subsequently create the graph data component if it is absent after running the plugin (described later).

The *content* property of graph data component is of *asset* type. This property type in WebGME supports values having complex data structures such as a file or a self-defined data in JavaScript. We use it to store a JavaScript dictionary with six STRIDE risks as keys and value to each key also a dictionary with key named *CAT* or *SAG*. The RAF plugin goes through the system and generates these

graphs and saves them to the graph data components in devices in a structured format. When the user switches to CAT or SAG visualizer, the visualizer reads the graph data component and renders the trees in WebGME.

3.4 Component Attack Tree

A Component Attack Tree (CAT) represents how vulnerabilities propagate inside a component. The *root* node of a CAT is one of the risks in the STRIDE category (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) [16]. The *leaf* nodes are the source causes of the risk at the root node. For example, the cause *memory access* implies that if the memory for running task of a component is accessible and modifiable externally, there can be some unusual behavior caused by unexpected memory access. Between the root node and leaf nodes are *intermediate* nodes, which represent internal behavior or phenomenon caused by other leaf nodes or intermediate nodes.

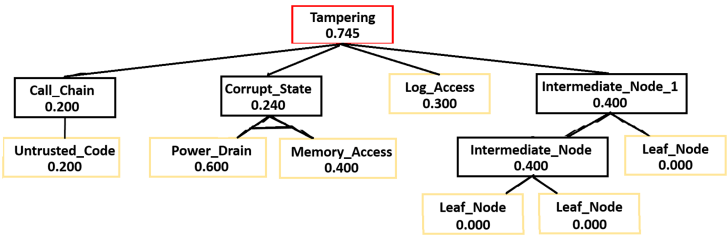

Fig. 2. Component attack tree for *Tampering* risk of a sensor

Each node contains a *vulnerability_score* property which represents the probability of exploitation and is given by domain experts that access known vulnerabilities and likelihood of those getting exploited. The vulnerability scores are propagated from bottom to top. The event of each node can occur for any combination of the occurrence of the events of its children nodes. By default, the children nodes are in an OR relationship with each other, which means if one of the children event occurs, the parent event will also occur.

Leaf nodes can contain, in addition to a vulnerability score, a mitigation score that represents a security mechanism that users put in place in order to mitigate the risk to a certain degree. For example, *Log_access* is a potential source of risk which can be a leaf node of a CAT. We can use some access control techniques, such as authentication or a privileges system, to prevent the malicious *log_access* behavior. The *mitigation_score* indicates how much these security mechanisms can prevent the vulnerability from propagating. Thus, the final vulnerability propagated to the component due to *log_access* behavior is given by: *vulnerability_score * (1 - mitigation_score).*

Figure 2 shows an example of a CAT for Tampering risk of a sensor in the case study system described in Sect. 4. The red block on the top is the objective

of this tree which is the risk itself. The children of the root node in black are intermediate nodes representing intermediate cause in one of the risk propagation paths. The leaf nodes in yellow represent the root causes of the risk.

We briefly described earlier how vulnerability scores are propagated. The *AND* logical relation among *Corrupt_State* and its children is shown by horizontal line. It implies that *Corrupt_State* can occur if and only if both *Power_Drain* and *Memory_Access* issues occur.

3.5 System Attack Graph

Similar to the CAT modeling process, the System Attack Graph (SAG) encompasses a root node, intermediary nodes, and leaf nodes. The root node corresponds to a target component STRIDE threat category, which also represents that component's CAT root node. An intermediary node represents a component *attack port*, which is used for propagating risk between components. As mentioned earlier, these attack ports are dependent on respective CAT root node risk levels, leading to the STRIDE threat categories represented by the CAT root nodes to be assigned as children leaf nodes of the attack ports. One difference in SAG from CAT is that the connections between nodes represent a path to reach a target instead of a hierarchical relationship.

When developing the SAG, the scoring assignment methodology is similar to the CAT. For the leaf nodes in the SAG, which are CAT root nodes, the score achieved from the CAT risk propagation process is used. Therefore, the score of the SAG leaf node should be the same as the CAT root node for the respective component STRIDE threat category.

Since the component attack ports do not have assigned risk scores, the intermediary nodes start as unassigned. Further, the root node also begins as unassigned. From this point, the risk from the SAG is propagated to the intermediary nodes until the root node has an assigned risk score. Next, the threat modeler can compare the SAG score for the root node to the relating CAT root node score to analyze whether the highest component threat is internal or via system-level propagation, and choose mitigation measures accordingly.

3.6 Algorithms for Vulnerability Propagation

The *EvaluateSystemLevelRisks* plugin is executed by clicking the play button in the top-left corner in WebGME and it affects the model only when executed under a Folder, Device, or a STRIDE risk component. In this section, we provide the key techniques in the plugin implementation (written in JavaScript). When this plugin is executed, WebGME creates an instance of this module and calls its *main* function. WebGME's *Core API* [4] is the major developer interface used for developing this plugin. The Core API provides the developer an interface for reading and modifying data in WebGME projects. The goal of the plugin execution is generating graph data for the visualizer. As described below, the code is organized into three parts: layered execution, reading and validation of the graph table, and generation of graphs.

Layered Execution. The *EvaluateSystemLevelRisks* plugin can be executed only inside a system folder, a device component, or a STRIDE risk within a device. Those three types have a containment relationship from top to bottom. A system folder contains devices, which contain STRIDE risks (given by domain experts based on known vulnerabilities). The execution also proceeds in a layered manner. When executed under a device, the plugin updates all the STRIDE risks under it. When executed under a system folder, the plugin collects all devices, recursively updates the devices, which in-turn updates all the STRIDE risks. We avoid repeated loading of graph data during accessing system components by putting the graph data component and its data in the function call so that the data is acquired only when it is undefined. The impact of exploitation is assessed through calculating system level risks (Sect. 3.8) and the system's safety is assessed through cyber gaming of exploitations and mitigations (Sect. 3.9).

Reading and Validating Graph Table. The graph table component under the system folder is the key for generating SAGs. When the plugin is executed, the graph table information is processed for graph generation. The plugin proceeds only when it finds no rule violations in the graph. If it finds violations, it warns the user showing the part of the graph containing errors.

The extracted data from the graph table is organized in a JavaScript dictionary. The graph table contains a directed graph of risk dependency. The keys in the dictionary are *Component ID* of the nodes under the graph table's crosscut, and the values are a list of component IDs that represent the out-degree nodes of the key's node. Since the dependency connections are a contained element of the graph table, the code iterates through all of the graph table's children, which are dependency connection components, and modifies the graph data dictionary.

Errors are collected in a list throughout the graph building process. At each dependency, the source and destination nodes are validated. Although even with a single error the SAG is not generated, the plugin still continues going through the whole dependency graph to detect and present all the errors in the model to the user for tracking the error sources. After iterating through the dependency connections and generating the graph's out-degree dictionary, the algorithm checks for any cycles in the graph.

Graphs Generation. The result of plugin execution is the CAT and SAG data. For each node in the graph, its children property lists the children trees. Both CATs and SAGs are stored in a JavaScript dictionary in the following form:

{*"name": <string>, "risk":<string>, children: [<graph dict>]*}

Both CAT and SAG are generated recursively. For CAT, the plugin first generates the CAT for all children, then calculates the risk value based on its children's risk values, puts children in the list, and then returns the tree dictionary of current node. The RAF metamodel has specific types for *intermediate* node and *leaf* node. The leaf of the CAT tree must be of type leaf node, which represents the source cause of internal risks. If a non-leaf node does result from any leaf node cause, in the recursion it will return a NULL. During recursion,

if a non-leaf node does not have children, or all children result in a NULL, the current return value will be a NULL and recursion will exit.

For SAG, the plugin generates a result dictionary from the graph table data. Dictionary data extracted from the graph table provides directed edge information. Starting from the root node, the SAG generation function recursively accesses out-degree node from the current node in the dependency graph. Similar to CAT, SAG only allows STRIDE risk as the leaf nodes of its tree structure and when recursion function is on an attack port, and if it does not have any child or all its children result in a NULL after recursion, the current return value will be NULL. The branch with attack port as leaf node will be cut off.

The CAT and SAG generated by the plugin are both tree-based structures with the same risk value propagation mechanism. In the tree, each node has a real numbered value ranging from 0 to 1 representing the probability of the risk occurrence. In both CAT and SAG, the root node is the final target for the whole tree, which is one of the STRIDE risks and is the ultimate goal of the vulnerability propagation.

3.7 Tree Visualization and Algorithms

RAF has separate visualizers for CATs and SAGs. These two visualizers differ slightly. In the sections below, we describe two major implementation issues with visualizers. The first is about handling events because the project information comes as asynchronous events and the code should handle them properly before rendering the graph. Secondly, the layout for the visualized graph must be correctly setup for efficiently and accurately rendering the graph's visual elements.

Event Handling. When switching to a visualizer, a *reload* event occurs in the browser. Upon switching the visualizer context, the client sends a request to the server for background data while refreshing the page. The requested data includes the current WebGME node and its children nodes. Loading the page content consists of several parts, each of which is processed in an asynchronous manner, and a callback is called when it is finished. Rendering cannot start as soon as the visualizer code is executed from the entry function because the key data may not be loaded by then. However, due to the asynchronicity of the loading event, we cannot determine which node is loaded last so that we can render the graph after that. Therefore, we designed a graph data component to store all the visualization information such that the visualizer only needs to load the graph data before rendering. The visualizer initializes the graph data variable as NULL. When a node is loaded, the code will render the graph only if the variable is assigned with node content. To avoid duplicated rendering, we use a flag indicating whether the graph is already rendered at current context.

Layout Method. The rendering objective is drawing a tree on a HyperText Markup Language (HTML) web page. HTML elements can be configured with width, height, and position. The key problem with the layout algorithm is how to

set up the position and size information of each node's elements. We truncate the scores as they are floating-point numbers and limit string names to three units in size. The exact position of a node is determined by not only its topological position in the tree, but also the position and size of other nodes. In the visualizer, we used a recursive function to generate another tree dictionary that records layout information for all nodes.

3.8 Risk Profile

To assess the system-level risk, the *EvaluateSystemLevelRisks* plugin updates the value of a risk profile component inside a system folder. This value represents the overall risk of the whole system and six STRIDE risk values along with their corresponding weights that contribute to the overall risk assessment score.

Each STRIDE risk values in the risk profile is calculated from all the corresponding risks of top-level devices as well as lower-level devices if they are under risk and there is a path by which the risk can be propagated to the logical upper-level system. The weight represents how each kind of risk contributes to the system-level risk and it can be customized by users. The risk weights are normalized during the computation. The overall risk value is calculated by combining all normalized risk values with OR logic.

In addition, we developed a tool that computes system-level risks when one of the device's risks is being exploited. When a device risk is ongoing, its risk value is temporarily changed to 1.0 and its sibling risks are turned to 0.0. The risks in other devices remain the same. Such change can have the effect on other risks through risk propagation and result in different system-level risk values, which show how vulnerable the system is when a device risk is actually being exploited. This can help to find the weakest points in the system. The tool rank orders the system vulnerabilities by sorting the corresponding system-level risk values in descending order. These risk analysis results can be downloaded after this tool is executed. RAF generates a hyperlink in the output to directly open the associated risk components that are listed in the above priority order.

3.9 Cyber Gaming for Risk Management

We also have a metamodel (not shown for brevity) for risk management plan modeling. An example risk management plan model is shown in Fig. 3. We can further connect our risk analysis model with attack behavior models to facilitate users evaluating the effectiveness of counterattack strategies. The ultimate goal of doing risk analysis is to make the system more secure. By implementing the risk profile features, we can spot the specific points that trigger the risk easily. But the actions available for us to mitigate vulnerabilities are limited. In addition, different mitigation actions (e.g., firewall, encryption, etc.) take time to implement and also incur a cost on performance of the system. Therefore, we must consider which mitigation actions could be applied and under what circumstances. In addition, certain mitigation actions may be kept secret from potential attackers and used only when absolutely needed. Attackers usually do

not just exploit one weakness in a device. Instead, a series of exploitations that interact with counterattack mitigations is used. Thus, it is important to model the attack and counterattack behaviors and analyze the risks step by step.

We developed the metamodel for risk management plans as a separate subgroup. The risk management plan allows us to model cyber gaming scenarios that allow combining and evaluating vulnerability exploitation against mitigation actions. The attack and counter-attacks can be modeled up to any depth.

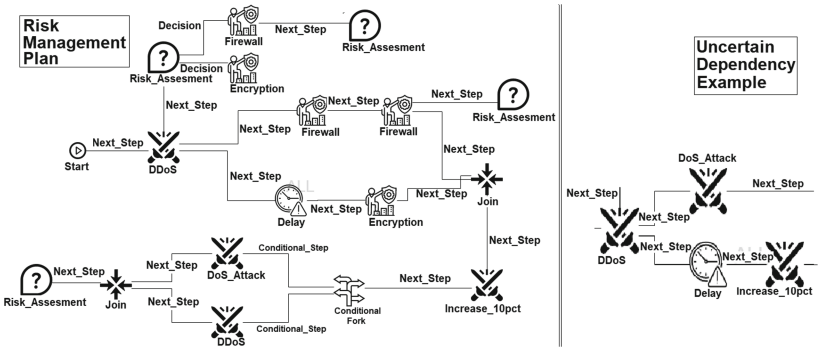

Fig. 3. Sample risk management plan with an example of uncertain dependency

The model represents a timed execution flow in which each process unit can take a certain amount of model time before its successors start running. There are two major types of processing units, exploitation and mitigation. They both associate with a risk component in the system architecture model via set relationship called *Risk_match*. An exploitation activates a certain risk by setting the vulnerability score to 1.0 and disabling the sibling risks under same device by setting their vulnerability score to 0. A mitigation reduces the chance of a device being exploited with a certain risk. Mitigation nodes have an attribute *mitigation_rate* that ranges from 0.0 to 1.0 and determines what percent of risk can the mitigation reduce. After a mitigation is processed, the vulnerability of the risk associated with the mitigation node will be reduced by the mitigation rate. In addition, RAF has a delay component that only delays the model time in the amount of specified time units. Since exploitation and mitigation are the simulation of an actual behavior, their processing time should have some associated variance. The *duration* attribute can be used to specify the time delay and *delay_variance* attribute can be used to specify the variance of the randomized processing times according to a Gaussian distribution.

Exploitation and mitigation components have a dependency limitation of what device they can be associated with. A device can be exploited with one of the risks only if any of its children is exploited. The device architecture is in a hierarchical structure – the nodes at the bottom are base devices and the others are high-level devices. At the beginning, only base devices are available

for exploitation. After a risk of a device is exploited, the device's parent device is available for further exploitation. The plugin will check for a possible dependency problem that one exploitation might be associated with an unavailable device at the time of execution. Because of the parallelism in the execution and uncertainty of the process time, we cannot determine whether an exploitation is associated with an available device at the time of execution. But we can determine whether an exploitation is guaranteed to be associated with an available device. On the right side of Fig. 3, the *DoS attack* (in the upper branch) exploits a risk in base device A, and an *Increase_10_percent attack* (in the lower branch) exploits device B (which is a parent of device A). Here, if none of the exploitations on B's children were finished, B's availability will be indeterminable when *Increase_10pct attack* starts. If *DoS attack* finishes before the *Increase_10_pct attack*, then B will become unavailable for *Increase_10pct attack*. However, if *DDoS attack* (predecessor of both *DoS attack* and *Increase_10_pct attack*), exploits one of the B's children C, but is not finished, then B will still be available for *Increase_10_pct attack*. The plugin checks for device availability before executing exploitations.

As the example shows, modeling of cyber-games allows parallel branches. From one process node, the model allows multiple next steps from it. Each parallel branch represents a time independent execution flow. A *join* node serves as a synchronizing point of parallel branches. Join nodes allow multiple input flows but there is only one output flow. Each input flow will suspend on the join point until all other input flows are finished. A *conditional fork* allows *random* selection of outgoing branches based on probabilities specified on them. When the sum of probabilities on all conditional branches does not equal to 1.0, the plugin normalizes all of the probabilities.

Risk assessment node calculates the overall risk at the time point it is assigned. The risk calculation is the same as the static calculation, but it also includes dynamic connections – a key feature of RAF that existing works mentioned in Sect. 2 do not support. In each physical (network) connection, there is an *active_condition* attribute that specifies when the connection is established in model time. The *active_condition* attribute is of string type and its value is an expression using the variable t such that it must evaluate to true or false. By default, the value for this attribute is *true*, which means the connection is always active. If the attribute is $t > 3$, for example, it means the connection is active when the model time is greater than 3. The expression can also incorporate periodic patterns such as $(t\%5 > 1)\ \&\ (t\%5 < 2)$, which means there is an active period with length of 1 in every 5 model time units. The overall risk value is calculated by propagating the risk through the dependency table. Each dependency is a directed connection between risks and attack ports that either relies on hierarchical relation or physical connection between the risks and ports. For those dependencies between two base devices that are connected with a physical connection, the status of this dependency depends on the active condition of the connection.

The *decision* branch after a risk assessment node uses a true/false expression to determine whether to continue with any following nodes (e.g., a mitigation).

The expression is written using the value of *variable name* attribute of previous risk assessment (which will represents the corresponding risk value). For example, if we set variable name as r, the expression on decision branches after the risk assessment can be $r > 0.1$, $1/r < 3$, etc. When expressions on multiple decision branches are true, the nodes following them are executed in parallel.

4 Case Study from a Railway Cyber Network

4.1 Railway System Model

There are two default visualizers at the top level folder of the example project, viz. *meta* and *composition*. *Meta* is a built-in visualizer that always shows the global metamodel design at any component folder of the system and is the default one when opening the project. The *composition* visualizer next to the *meta* visualizer shows the model components, as shown in Fig. 4.

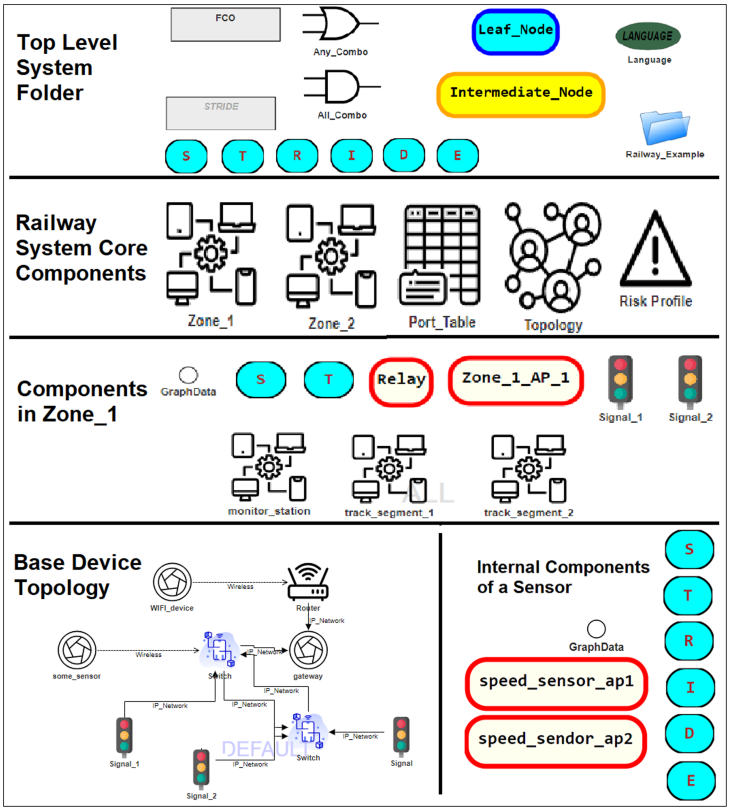

Fig. 4. Railway system example model

In the top folder of the main project, the component with a blue folder icon (named *Railway_Example*) is an example Railway Transportation System built using the metamodel described earlier. This a very simple model of a railway control system with a few sensors placed on the railway track to collect real-time signal data, a repeater for receiving, filtering, and transmitting the raw data, a gateway and a router as parts of the network, and a central station that receives all data globally and sends out control commands. The directed lines with arrows between devices show the connection from one device to another, with type names along with the line. In the example system, all connections are bidirectional, so there are always arrows at the end of lines between devices in its composition. There is also a graph table which is unique to the system and used to specify the risk propagation dependencies between attack ports and STRIDE risk components. Each type of component is assigned with an SVG icon to its metamodel. Figure 4 shows the composition of the example system, the constituents inside *Zone_1* device of the top folder, and the topology of base devices under the crosscut panel of Topology component in the system.

When double-clicking on a device component (e.g., speed_sensor) for example, the composition will show the internal component of this device, as shown in Fig. 4. The blue boxes with letters on them are STRIDE risk components, which contain a CAT organized with the folder hierarchy which will be further discussed below. The white components with red borders are attack ports. Attack ports represent certain attack behavior that can interact with an external device. For example, sending malicious packets can be an attack port of the central station because the central stations are able to send packets that may possibly cause unexpected behavior and can cause risk to the system if the packets are somehow sent with a malicious intention. If malicious packets are sent, that may further cause integrity problems in a repeater, or disrupted communication of a router or gateway, and finally result in one of the STRIDE risks in one of the devices. Such risk propagation is modeled and captured by SAG, that is described below. There is also a small circle component called *Graph Data*. This is used to store the results of computations performed by the plugin and read by visualizers to show the risk analysis graphs. It is not intended to be used by users directly.

When the user enters into one of the STRIDE risk components in a device, there may be a few children components that are in the CAT under the root node. The visualizers work for STRIDE components here and enable the *ComponentAttackTree* and *SystemAttackGraph* panels inside the STRIDE component. If the user switches to one of these visualizers, the visualized tree structure for the corresponding structure will be displayed. Figure 2 shows the CAT of *Tampering* risk of a sensor and Fig. 5 shows its SAG.

Another component is the *Graph Table*, with table like SVG icon. On selecting it, the composition view (empty) is shown, by default. However, if the user switches to the visualizer *crosscut*, a graph is shown with blue *STRIDE* components and red *attack port* components and have a directional edge between the nodes (see Fig. 5). This graph is defined by the user and it summarizes the paths of how external risks can propagate from one component to the other. The nodes of a crosscut are not contained in the graph table component but they are

simply references to devices in the system folder. The connection components are included by the parent component but they are not shown in the composition due to inherited containment relationship to the Graph Table model.

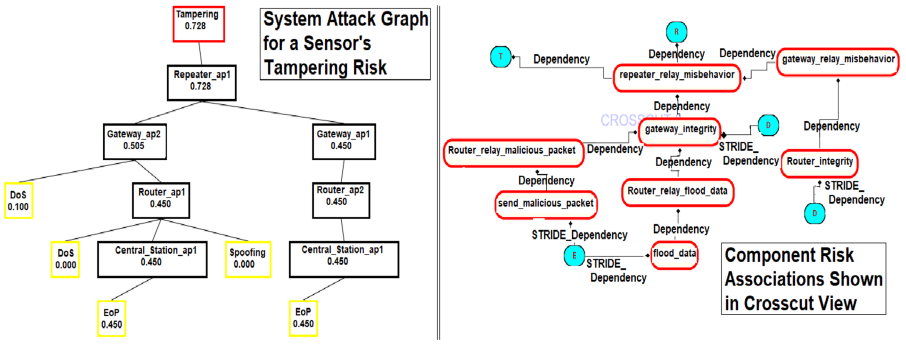

Fig. 5. SAT and risk associations

4.2 Modeling the System

To create our own model based on the project, we need to first create a Project Folder component at the top-level hierarchy of the main project. The name of a component is the metamodel's name, so it is important to rename the system folder component with a proper name for the system after creating it.

In the system folder, we can build the system topology with device components and connections. On the left there is an area with all available components in the current context are shown. Dragging an icon from the left panel to the white space can create the corresponding component. Dragging lines between device components can create connections between them. There is a prompt choice for the type of connection after drawing a line from one device to another.

Inside a STRIDE risk component, we can put intermediate nodes or leaf nodes. Those nodes are constituents of the CAT of the parent STRIDE risk device. The risk nodes inside a STRIDE risk component are organized in a tree structure based on their hierarchical relations. For each node, its children nodes are the direct causes of it. The leaf nodes require the user to input the risk values based on their domain knowledge and optionally fill the property of mitigation information.

The *EvaluateSystemLevelRiskss* plugin can be executed under three types of components: project folder, device, and STRIDE risk component. These three types of components have hierarchical relations. A project folder contains devices and a device contains STRIDE risks. When the plugin is executed under the project folder, all devices and the risks under them will be updated. If the plugin runs under a device, only the risks under the current device will be updated. If the current running environment is just one of the STRIDE risks, only the information related to this risk will be renewed. Updating globally can take a

while if the system is large. If we need the result for only part of the system or if we modify only a part of the system before the update, local updates (device, risk level) can be used.

4.3 Risk Dependencies

A graph table uses the *crosscut* visualizer instead of the *composition* visualizer to model dependency relations between STRIDE risks and attack ports of various devices. In *composition*, all elements shown in the white space are direct children of the current element being inspected, and connections are created by dragging from one component to another and permissible only between nodes with a common parent. In RAF, crosscuts are used for modeling risk propagation dependencies across component hierarchies by creating connections between references of existing nodes.

The direct graph under the crosscut of a graph table must be checked for validity, so the RAF's plugin first checks that. There are three rules for the graph validity. First, there should not a cycle. The graph is built for summarizing risk dependencies and the directed edges are a representation of the potential path for risk propagation. A cycle would be logically incorrect because a risk should have its ultimate source and destination in the graph. Even if there are cycles due to the device property, the user assigning the dependency should resolve it using domain knowledge. The dependency can be from an attack port to another attack port, an attack port to a STRIDE risk, and a STRIDE risk to an attack port. For the attack port to attack port dependency, the attack ports must be from different components. For the dependency from STRIDE risk to an attack port, they cannot be within the same device. For the dependency from attack port to a STRIDE risk, they must be within the same device. This is because the graph and dependency relations are used for external risk analysis. Risk occurrence in each device is triggered by factors in other devices when the attack port is either triggered by another attack port, or internally by a risk within the device. Any errors in graph model are reported with locations.

4.4 Risk Management Plan

Figure 3 shows a full example of the risk management model. Similar to modeling the system architecture and risk node, the components are simply added from the left panel and connected by drawing from one point to the other.

From the start node, the first node is a DDoS attack and it will result in spoofing risk of one of the traffic lights. Figure 6 shows the topology associated to this example risk management plan. The device that first DDoS attacks is *Signal_1* marked by a red circle and it has an IP network connection with a switch. The active condition attribute for the marked connection from *Signal_1* to the switch is $t > 1$ & $t < 2$ which means the connection is only active during the time interval (1, 2). Figure 6 shows a part of the port table associated with the risk management plan. The spoofing risk marked with a red circle is the exploited spoofing risk of *Signal_1*. The first relay is the relay inside *Signal_1*

and the second relay after it is in the switch. *Signal_1* and the switch are both base devices, so the dependency between two relays is built upon this conditional connection. Such dependency inherits the condition of the connection and the dependency is also *active* when the connection is active. In this example, the tampering and spoofing risks only propagate to the top during the time interval (1, 2).

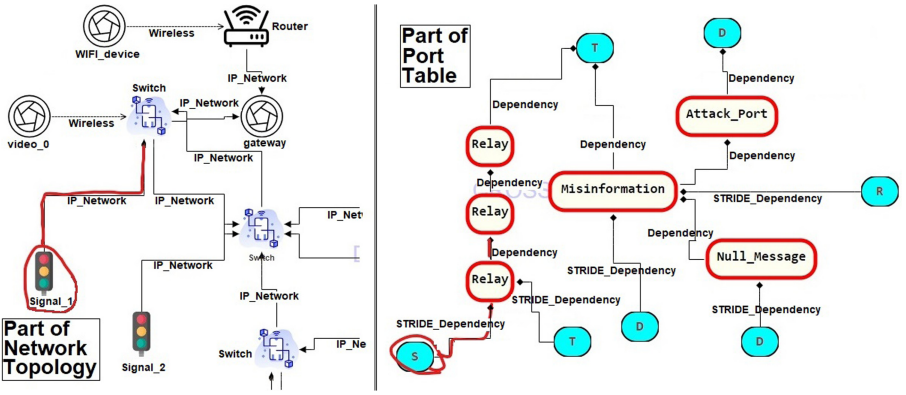

Fig. 6. Part of the network topology and port table

The risk assessment followed by next steps of mitigation nodes shows an example of the conditional action based on risk assessment result. The variable name specified in the risk assessment is *risk_var*. The first branch has the expression *risk_var >0.5 & risk_var <0.6* and second branch has *(risk_var ** 2) >0.3*. Those branches will proceed if the expression is true. In this case, the execution path is not determined before each run because there is uncertainty of the risk assessment value. The connection between *Signal_1* and the switch is only active during time interval (1,2) and whether it is active or not has impact on the overall risk value since there is a risk propagation dependency that is established upon the connection. On the other side, the actual running of first DDoS is uncertain because the actual time the exploitation takes is randomly assigned based on a normal distribution with given mean and variation. If we let the expected execution time of the DDoS exploitation be 1, the actual finish time could be somewhere around 1.0 (e.g., 0.99 or 1.01). If the risk assessment takes place at 0.99, the important dependency that propagates that risk is inactive at this time point and the exploitation will have no effect on the overall risk. Conversely, if the risk assessment runs the risk propagation at the time point at 1.01, the exploited spoofing will be propagated to the top and increase the risk value. Therefore, the action of the next step will also be random due to the randomness of risk assessment result.

After parallel branches merges at the join point, there is a conditional fork node after it. The connections that follow the conditional fork are of type *conditional step* and have a probability attribute. The sum of probabilities from

the fork point should be 1.0. However, when the user enters probabilities that do not add up to 1.0, the plugin will normalize the values. Only one branch is selected during the execution, and the selection is a weighted random based on probabilities of active branches.

5 Conclusion and Future Work

In this paper, we demonstrated the use of the *model-integrated computing (MIC)* technique to build a metamodel for risk analysis and demonstrated it using a railway transportation system case study. Our web-based Risk Analysis Framework (RAF) provides a set of risk analysis and visualization tools that aim to provide a user-friendly platform for risk analysis and risk management planning.

After modeling the system architecture and network topology, risk analysis tools are used for automatically propagating the risk values, and visualization tools are used to visually inspect the component attack trees of specific component risks and system attack graphs that also consider how risks can propagate across components via their network interconnections. In addition, our automated risk assessment tool can analyze all the modeled system vulnerabilities, evaluate them for their impact on overall system-level risk, and rank order them for targeting mitigation actions against most damaging vulnerabilities. We also provided a novel approach to handle dynamic network connections for analyzing risks amidst changing network topology of infrastructure components, such as mobile locomotives and its on-board devices in the railway transportation systems. The quantitative approach to risk analysis and model-based design and automated analysis tools provides a highly powerful framework for analyzing risks of critical infrastructures, such as a railway transportation system.

It is important to note that the algorithms and approaches developed in this work are equally applicable to other types of critical infrastructures. We are currently working on applying it to energy, water distribution, and healthcare domains. Also, we are working on integrating the risk analysis framework with the networked co-simulations of dynamical systems (e.g., Cyber-Physical Systems Wind Tunnel (CPSWT) [17,18]) for validating and improving risk scores as well as for conducting informed, simulation-based cybersecurity evaluations.

Acknowledgement. This work is supported by the US National Security Agency (NSA) (award #H98230-18-D-0010) and the US National Institute of Standards and Technology (NIST) (award #70NANB20H020). No approval or endorsement of any commercial product by NSA or NIST is intended or implied. Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by NSA or NIST, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

References

1. Koutsoukos, X., et al.: Performance evaluation of secure industrial control system design: a railway control system case study. In: Resilience Week, pp. 101–108 (2016)
2. Myagmar, S., Lee, A.J., Yurcik, W.: Threat modeling as a basis for security requirements. In: Symposium on Requirements Engineering for Information Security (SREIS), vol. 2005, pp. 1–8 (2005)
3. Sztipanovits, J., Karsai, G.: Model-integrated computing. Computer **30**(4), 110–111 (1997)
4. Kecskes, T., Zhang, Q., Sztipanovits, J.: Bridging engineering and formal modeling: WebGME and formula integration. Technical report in Department of EECS, Vanderbilt University, Nashville, TN (2017)
5. Liao, Y.-C.: Quantitative information security vulnerability assessment for norwegian critical infrastructure. In: Rashid, A., Popov, P. (eds.) CRITIS 2020. LNCS, vol. 12332, pp. 31–43. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58295-1_3
6. Schmittner, C., et al.: Threat modeling in the railway domain. In: Collart-Dutilleul, S., Lecomte, T., Romanovsky, A. (eds.) RSSRail 2019. LNCS, vol. 11495, pp. 261–271. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-18744-6_17
7. Saini, V., Duan, Q., Paruchuri, V.: Threat modeling using attack trees. J. Comput. Sci. Coll. **23**(4), 124–131 (2008)
8. Martins, G., Bhatia, S., Koutsoukos, X., Stouffer, K., Tang, C., Candell, R.: Towards a systematic threat modeling approach for cyber-physical systems. In: Resilience Week (RWS 2015), pp. 1–6. IEEE (2015)
9. Kavallieratos, G., Spathoulas, G., Katsikas, S.: Cyber risk propagation and optimal selection of cybersecurity controls for complex cyber-physical systems. Sensors **21**(5), 1691 (2021)
10. Stellios, I., Kotzanikolaou, P., Grigoriadis, C.: Assessing IoT enabled cyber-physical attack paths against critical systems. Comput. Secur. **107**, 102316 (2021)
11. Garg, U., Sikka, G., Awasthi, L.K.: Empirical analysis of attack graphs for mitigating critical paths and vulnerabilities. Comput. Secur. **77**, 349–359 (2018)
12. Liu, X., Zhang, J., Zhu, P., Tan, Q., Yin, W.: Quantitative cyber-physical security analysis methodology for industrial control systems based on incomplete information Bayesian game. Comput. Secur. **102**, 102138 (2021)
13. Shostack, A.: Threat Modeling: Designing for Security. Wiley, Hoboken (2014)
14. Potteiger, B., Martins, G., Koutsoukos, X.: Software and attack centric integrated threat modeling for quantitative risk assessment. In: Proceedings of the Symposium and Bootcamp on the Science of Security, pp. 99–108 (2016)
15. Garitselov, O., Mohanty, S.P., Kougianos, E.: A comparative study of metamodels for fast and accurate simulation of nano-CMOS circuits. IEEE Trans. Semicond. Manuf. **25**(1), 26–36 (2011)
16. Microsoft Security Development Lifecycle (SDL) Threat Modeling Tool. https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool. Accessed 27 Aug 2021
17. Neema, H., Sztipanovits, J., Steinbrink, C., Raub, T., Cornelsen, B., Lehnhoff, S.: Simulation integration platforms for cyber-physical systems. In: Proceedings of the Workshop on Design Automation for CPS and IoT, pp. 10–19 (2019)
18. Neema, H.: Large-scale integration of heterogeneous simulations. Ph.D. dissertation Research. Vanderbilt University (2018)