Methodology for Resiliency Analysis of Mission-Critical Systems

Mahmoud Abdelgawad
m.abdelgawad@colostate.edu
Department of Computer Science,
Colorado State University
Fort Collins, Colorado, USA

Indrakshi Ray
indrakshi.ray@colostate.edu
Department of Computer Science,
Colorado State University
Fort Collins, Colorado, USA

ABSTRACT

Mission-critical systems ensure the safety and security of any nation. Attacks on mission-critical systems can have devastating consequences. We need to design missions that can prevent, detect, survive, recover, and respond to faults and cyber attacks. In other words, we must design missions that are cyber-resilient. System engineering techniques must be used to specify, analyze, and understand where adverse events are possible and how to mitigate them while a mission-critical system is deployed. This work introduces an end-to-end methodology for designing cyber-resilient mission-critical systems.

The methodology first specifies a mission in the form of a workflow. It then converts the mission workflow into formal representation using Coloured Petri Nets (CPN). The methodology also derives threat models from the mission specification. The threat models are used to form a formal specification of attacks that can be represented in CPN. These CPN attacks are plugged into potential places in the CPN mission to design various attack scenarios. The methodology finally verifies the state transitions of the CPN mission attached to attacks to analyze the resiliency of the mission. It identifies in which state transition the mission succeeds, fails, and is incomplete. The methodology is applied to a drone surveillance system as a motivating example. The result shows that the methodology is practical for resiliency analysis of mission-critical systems. The methodology demonstrates how to restrict a mission to improve the resiliency of mission-critical systems. The methodology provides crucial insights in the early stages of mission specification to achieve cyber resiliency.

CCS CONCEPTS

• **Applied computing** → **Cyberwarfare**; • **Software and its engineering** → **Formal software verification**; **Petri nets**.

KEYWORDS

Mission-critical Systems, Resiliency, Workflow, Formal Methods, Coloured Petri Nets (CPN)

ACM Reference Format:
Mahmoud Abdelgawad and Indrakshi Ray. 2024. Methodology for Resiliency Analysis of Mission-Critical Systems. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3605098.3636066

SAC '24, April 8–12, 2024, Avila, Spain

ACM ISBN 979-8-4007-0243-3/24/04.
https://doi.org/10.1145/3605098.3636066

1 INTRODUCTION

Cyber resiliency is conceptually identical to cyber survivability defined for warfighter systems [13]. It focuses on evaluating cyber resilience goals, including anticipating, withstanding, recovering from, and adapting to adverse cyber events that could impact mission-related functions.

A mission-critical system is one whose failure significantly impacts the mission-related functions [5, 11]. Examples of mission-critical systems include navigational systems for a spacecraft and drone surveillance systems for military purposes. These systems are prone to adverse cyber events and physical attacks because they can cripple a nation [2]. Mission-critical systems must fulfill survivability requirements to continue a mission in the face of attacks. Thus, cyber resilience engineering must specify and analyze a mission before deployment to evaluate its resilience and gauge what failures can be tolerated.

A mission can be described in the form of a workflow consisting of various tasks executed by subjects and connected via different types of control-flow operators [15]. The literature on mission resiliency introduces solutions to address the unavailability in the context of assigning subjects to tasks [9, 10, 14]. Subjects have attributes that provide them the capabilities to perform tasks. However, adverse cyber events and physical attackers can disrupt the capabilities of mission-critical systems, rendering performance degradation. The performance degradation may cause a mission to abort or fulfill only a subset of its objectives. In other words, attackers do not necessarily remove mission-critical systems from service permanently; they degrade their capabilities, but these systems may be able to continue to complete the mission.

Many types of workflow resilience are defined: static, decremental, and dynamic resilience [14]. Static resilience refers to a situation in which subjects become unavailable before the workflow executes, and no subjects may become available during the execution. Decremental resiliency expresses a situation where subjects become unavailable before or during the execution of the workflow, and no previously unavailable subjects may become available during execution, while dynamic resilience describes the situation where subjects may become unavailable at any time; a previously unavailable subjects may become available at any time. The different types of attacks necessitate having various notions of resiliency.

Analyzing workflows manually to check for resiliency is tedious, error-prone, and labour intensive. Towards this end, there is a need for automated analysis of a mission and evaluating all potential attacks and their effects. We use Coloured Petri Nets [6] for the purpose of analysis.

In this paper, we introduce a methodology that systematically analyzes various aspects of mission resiliency for mission-critical systems. The methodology leverages the workflow definition to

describe a mission. It also leverages a set of transformation rules [1] to convert the workflow to Coloured Petri Nets (CPN). We use CPN because it is suitable for modeling process interaction and provides primitives for defining data types (i.e., colors). It is also associated with a high-level programming language (i.e., CPN-ML [8]) and Integrated Development Environment (i.e., CPN Tools [12]) that are practical to simulate processes interaction, manipulate data values, and verify state transitions of systems.

The methodology starts with specifying a mission, analyzing it, and formally presenting it as a mission workflow. It uses specification analysis to derive threat models. The mission workflow is converted into the formal specification as a CPN mission. The threat models are used to form a formal specification of attacks as CPN attacks. These CPN attacks are used to demonstrate various attack scenarios. The methodology then verifies the state transitions of the CPN mission attached to attack scenarios. It analyzes the resiliency of the mission and identifies in which state transition the mission succeeds, fails, and is incomplete.

The methodology is applied to a drone surveillance system that performs a data-collection mission. The application of our methodology to the drone surveillance system covers decremental resiliency where the drone is attacked at the regions where it has to perform a surveillance mission and collect meaningful data.

We analyzed two versions of decremental resiliency: one-shot and decremental resiliency. One-shot resiliency refers to only one drone's capability being attacked [4, 14], and decremental resiliency demonstrate that various attacks degrade many capabilities. The application shows that the methodology is practical for resiliency analysis of mission-critical systems. The methodology helps to improve mission-critical system resilience.

We proceed with this paper as follows. Section 2 describes a surveillance drone mission as a motivating example. Section 3 presents our methodology of resiliency analysis for mission-critical systems. Section 4 applies the first 3 steps of the methodology to convert the drone surveillance specification. Section 5 applies steps 4 and 5 to describe the threat model and attack representations. Section 6 pursues applying the final 2 steps of the methodology to analyze drone surveillance and its resiliency. Section 7 summarizes some related work. Section 8 concludes the paper and points to future directions.

2 MOTIVATING EXAMPLE

In this section, we briefly describe an example mission using a surveillance drone example. The surveillance drone mission is to collect data over a region of interest.

The drone has a camera that can take pictures (i.e., scanning) at high and low altitudes and sensors capturing (i.e., measuring) heat signals and radiation levels. The sensors are only accurate at low altitudes. There are three regions of interest: Regions A, B, and C. Region A is where the drone is regularly scheduled to perform surveillance. This region is large and close to the deployment point (base). Regions B and C are smaller but are further away from the base. It is likely to be detected at low altitudes in Region A. Therefore, the drone can only fly over Region A at high altitudes, from where it can only use its camera to scan buildings and construction sites. If the drone is instructed to fly over Regions B or C, it can fly at high or low altitudes. However, due to the lack of visibility over Regions B and C, only the heat and radiation sensors can capture meaningful data. Thus, the drone can only collect data with its sensors over Regions B and C.

The drone keeps flying as long as the battery is enough to return to the base. We assume the battery level contains 4 units. Flying to Region A, B, or C takes 1 battery unit. Returning to the base also takes 1 unit. Actuating the camera and sensors each take 1 unit. The drone's status is checked (camera is on, sensors are on, and the battery is full) during deployment before flying to accomplish the mission. The mission is considered a success if the drone collects data and returns to the base.

Now, we need to analyze whether it is possible for the mission to succeed and in which attack scenarios the mission fails. An attack scenario changes the drone attributes, which can be attacked, and renders it unable to perform a task. For instance, an attack scenario defuses the drone camera and degrades its scanning capability. The attack scenarios must cover all possible drone degradation. A mission is resilient to an attack scenario if, after the attack occurs, it still meets its objectives.

3 METHODOLOGY

This section illustrates a methodology for analyzing mission resiliency for any mission-critical system. As illustrated in Figure 1, the methodology consists of seven steps, starting from a description of mission specification as input and ending with mission resiliency analysis. Steps 1, 2, and 3 are processed sequentially, as one step's output is the input for the next step. Step 4 gets the output from Step 1, processes it, and passes the outcome to Step 5. Notice that the outcome from steps 3 and 5 are required to process step 6. The iteration symbol (three arrows circle-shaped) expresses that the outcome from step 5 is processed many times. For every attack attachment designed in step 5, steps 6 and 7 are processed. The 7 steps are described as follows.

- **Step 1 - Description of Mission:** It is an essential step to define the mission *subjects* as active entities that execute tasks. Each *subject* has a type and a set of typed variables that represents *attributes* corresponding to its properties. The values of the attributes of a subject constitute its state. The state of a subject determines whether it can execute a given task. Step 1 also defines a set of *tasks*, which are atomic units of the mission. Tasks are performed as a control flow to form the entire mission. The mission always starts with an initial task and ends with a final task. Between the initial and final tasks, a mission includes a set of intermediate tasks that may be grouped into sub-sets of tasks, referred to as sub-workflow in the formal representation. A mission also has a set of *objectives* to be accomplished by subjects. If all *objectives* are achieved, the mission is *succeeded*. If some of *objectives* are achieved but not all of them, the mission is *incomplete*. When no objectives are satisfied, then the mission is *failed*.

- **Step 2 - Formal Representation of Mission:** This step formalizes the mission as a workflow. It formalizes the mission into many sub-workflows and connects them into a main workflow to represent the entire mission. It then instantiates

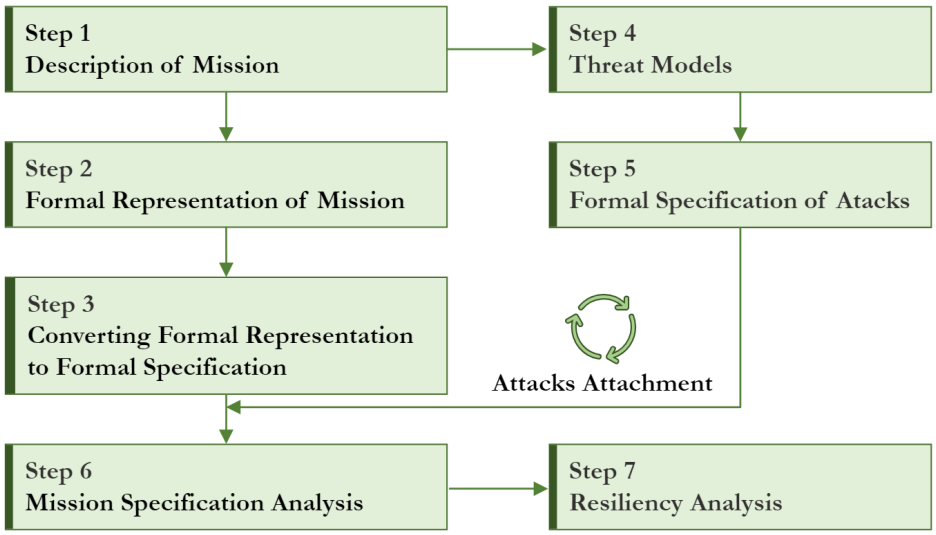

Figure 1: Methodology for Resiliency Analysis

the workflow. It instantiates each subject and its attributes, assigns each subject to tasks, and assigns initial conditions for the subjects and objectives.

- **Step 3 - Converting Formal Representation to Formal Specification:** The mission workflow generated by Step 2 is transformed into a formal specification. The formal specification is beneficial for verifying the correctness of the mission and analyzing its resiliency.

- **Step 4 - Threat Models:** The threat model is derived directly from the mission specification analysis. Step 4 uses each subject's *attributes* that are mutable and states the possible changes of these *attributes*. It also expresses the *impact* to the mission if an *attribute* is changed by attack.

- **Step 5 - Formal Specification of Attacks:** An attack is an action taken by an adversary that changes the state of a subject in a way that renders it unable to perform a task it has been assigned. Each attack described in the threat model is designed as a formal specification. An attack requires preconditions and postconditions to be inserted into the formal specification of the mission. Step 5 defines the preconditions and postconditions of the attacks and the changes on the subjects these attacks can perform.

- **Step 6 - Mission Specification Analysis:** This step executes iteratively as the number of attack scenarios would be attached to the mission specification. For every attack scenario attached to the mission specification, the state space of the mission specification is generated, and mission state transitions are verified and analyzed.

- **Step 7 - Resiliency Analysis:** Various types of mission resiliency can be analyzed, including static, decremental, and dynamic resiliency [14]. The resiliency analysis highlights the constraints that restrict the mission. These constraints improve the mission's resiliency.

4 MISSION SPECIFICATION TO FORMAL REPRESENTATION

This section applies the first two steps of the methodology to the surveillance drone to translate the mission described in section 2 into formal specification.

4.1 Surveillance Drone Mission Description

The specification of the surveillance drone mission is extracted from the example description (section 2). The surveillance drone mission specification is summarized as:
The set of subjects = {*drone*}, and the set of attributes = {*location*, *fly_enabled*, *battery_level*, *instruction_issued*, *camera_enabled*, *sensors_enabled*, *data_collected*}.
The tasks are:

(1) check the drone *status*: *camera is enabled*, *sensors are enabled*, and *battery is set to 4 units*.
(2) If the drone is *not instructed*, *fly to Region A* to scan buildings and construction sites.
(3) If the drone is *instructed*, *fly to Region B or C* to measure heat signals and radiation levels.
(4) *Return back to base* after *data is collected*.

The objectives = {*collect data*, *return to base*}. Notice that we italicized the mission elements, which are used to formalize the drone mission representation.

4.2 Formal Representation of the Drone Mission

We use the workflow defined in [1] to represent a mission. A workflow consists of tasks connected through operators. The syntax of the workflow is defined as follows.

Definition 4.1 (Workflow). A workflow is defined recursively as:

$$W = t_i \otimes t | W_1 \otimes W_2 | W_1 \# W_2 | W_1 \& W_2 | if\{C\}\, W_1\, else\, W_2 | \\ while\{C\}\{W_1\} \otimes t_f$$

where sequence, exclusive choice, and, conditioning, and iteration operator

- t is a subject-defined atomic task.
- t_i and t_f are unique initial and final tasks, respectively.
- $\otimes$ denotes the *sequence operator*. $W_1 \otimes W_2$ specifies W_2 is executed after W_1 completes.
- # denotes the *exclusive choice operator*. $W_1 \# W_2$ specifies that either W_1 executes or W_2 executes but not both.
- & denotes the *and operator*. $W_1 \& W_2$ specifies that both W_1 and W_2 must finish executing before the next task can start.
- $if\{C\}\, W_1\, else\, W_2$ denotes the *conditioning operator*. C is a Boolean-valued expression. Either W_1 or W_2 execute based on the result of evaluating C but not both.
- $while\{C\}\{W_1\}$ denotes *iteration operator*. If C evaluates to true, W_1 executes repeatedly until the expression C evaluates to false.

A mission is defined as one or many subjects performing tasks to accomplish objectives.

Definition 4.2 (Mission). A mission is defined as 5-tuple $\mathcal{M} = \left(W, \mathcal{S}, \mathcal{ST}, I, O\right)$ where W is the workflow corresponding to the mission, $\mathcal{S}$ is a set of subjects, $\mathcal{ST} \subseteq S \times Tasks\left(W\right)$ is the set of subject to task assignments, I is the set initial conditions, and O is the set of mission objectives. The conditions and objectives are expressed in predicate logic.

The surveillance drone mission is divided into deploy sub-workflow, Scan Region A, and Measure Region B/C sub-workflow. These three sub-workflows connect to the main workflow, which starts with the initial task and ends with the final task. Each sub-workflow includes a set of tasks managed by operators. The operators are also used to connect sub-workflows to the main workflow. The workflow of surveillance drone mission is described as:

$$W = Init \otimes Check\_Status \otimes Deploy \otimes if\left(instruction\right) \\ \{\left(Fly\_to\_Region_B \,\#\, Fly\_to\_Region_C\right) \otimes \\ \left(Measure\_Radiation\_level \,\&\, Measure\_Heat\_Signal\right)\} \\ else\{Fly\_to\_RegionA \otimes while\{battery\_level > 1\} \\ \{Scan\_Buildings \,\&\, Scan\_Constructions\}\} \otimes Return\_to\_Base \otimes Final$$

The workflow diagram, Figure 2, illustrates the surveillance drone mission. The dotted boxes represents the sub-workflows. It shows that checking the drone status and deployment (giving the drone instructions on where to fly) is a sub-workflow. The second sub-workflow is to fly to Region A and perform the scan tasks, and the third is to fly to Region B/C and perform the measure signals tasks. These three sub-workflows connect to the main workflow, which starts with the initial and final tasks. The *return to base* task can be part of the main workflow since it does not represent any sub-workflow. The visualization of the workflow helps ensure the correctness of the workflow transformation to the CPN; hence, it is useful. Now, we instantiate the workflow. We initialize each subject and its attributes, assign each subject to a set of tasks, and set initial conditions for the subjects and objectives. There is only one subject type *Drone*, and is instantiated as $\mathcal{S} = \{drone_1 : Drone\}$. The subject type *Drone* has a set of attributes as:
$Drone.Attributes = \{type : string; location : string;$
$fly\_enabled : bool; battery\_level \in \{1,2,3,4\};$
$instruction\_issued : bool; camera\_enabled : bool;$
$sensors\_enabled : bool; data\_collected : bool\}$
The subject $drone_1$ is assigned to every task as:
$\mathcal{ST} = \{\left(drone_1, transition\right) \mid t \in Tasks\left(W\right)\}$.

Let I be a predicate logic formula giving the initialization conditions as:
$I = \exists\, s : S \mid \left(type\left(s\right) = Drone\right) \wedge \left(location\left(s\right) = \text{“}base\text{”}\right) \wedge$
$\left(fly\_enabled\left(s\right) = true\right) \wedge \left(battery\_level\left(s\right) = 4\right) \wedge$
$\left(instruction\_issued\left(s\right) = false\right) \wedge$
$\left(camera\_enabled\left(s\right) = true\right) \wedge$
$\left(sensors\_enabled\left(s\right) = true\right) \wedge$
$\left(data\_collected = false\right)$.

Let O be a predicate logic formula that defines the mission objectives as:
$O = \exists\, s : S \mid \left(type\left(s\right) = Drone\right) \wedge$
$\left(location\left(s\right) = \text{“}base\text{”}\right) \wedge \left(data\_collected\left(s\right) = true\right)$.

As a result, the surveillance drone mission specification is described as: $\mathcal{M}_{drone} = \left(W, \mathcal{S}, \mathcal{ST}, I, O\right)$.

4.3 Converting Formal Representation to Formal Specification

We consider CPN as a formal model that is practical to express mission state transitions due to CPN providing color sets, a flexible way to define data types to represent various types of subjects [6, 7]. A CPN is a directed bipartite graph where nodes correspond to places P, transitions T, and Arcs A are directed edges from a place to a transition or a transition to a place. The input place of transition is where a directed arc exists between the place and the transition. CPN operates on multisets of typed objects called tokens. Places are assigned tokens at initialization. Transitions consume tokens from their input places, perform some action, and output tokens on their output places. The distribution of tokens over the places of the CPN defines the states, referred to as markings. Transitioning from one marking to another represents the system's state transition (i.e., state space). The CPN is formally defined as:

Definition 4.3 (Coloured Petri Nets (CPN)). CPN is defined as 9-tuple $CPN = \left(P, T, A, \Sigma, V, C, G, E, I\right)$, where P, T, A, Σ, and V, are sets of places, transitions, arcs, colors, and variables, respectively. C, G, E, and I are functions that assign colors to places, guard expressions to transitions, arc expressions to arcs, and tokens at initialization expression, respectively.

We use transformation rules defined in [1] to convert the mission workflow tuple into a CPN tuple as:

$$\mathcal{M} = \left(W, \mathcal{S}, \mathcal{ST}, I, O\right) \xrightarrow[Rules]{Trans.} CPN = \left(P, T, A, \Sigma, V, C, G, E, I\right)$$

These transformation rules are explained briefly by applying them to surveillance drone mission workflow:

Rule 1: Convert each task into a CPN transition.

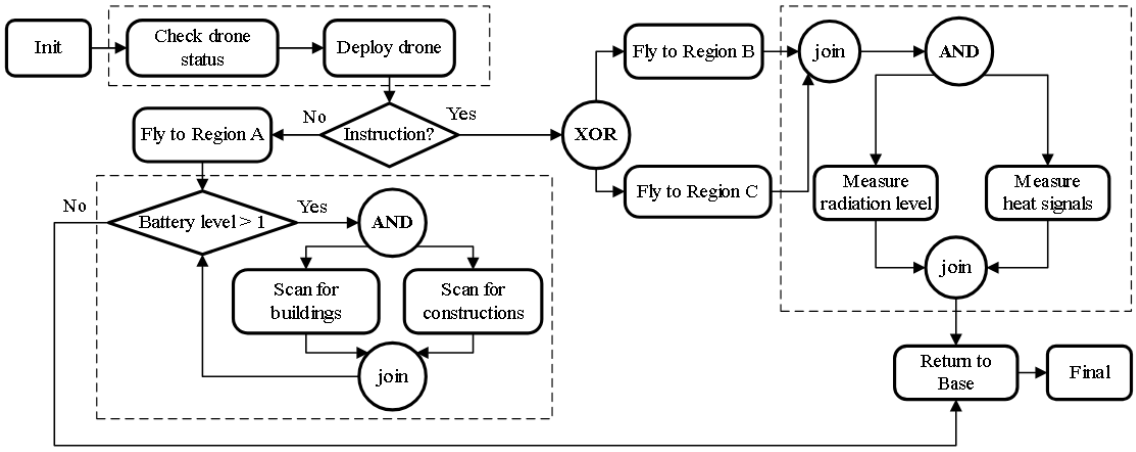

Figure 2: Surveillance Drone Mission Workflow

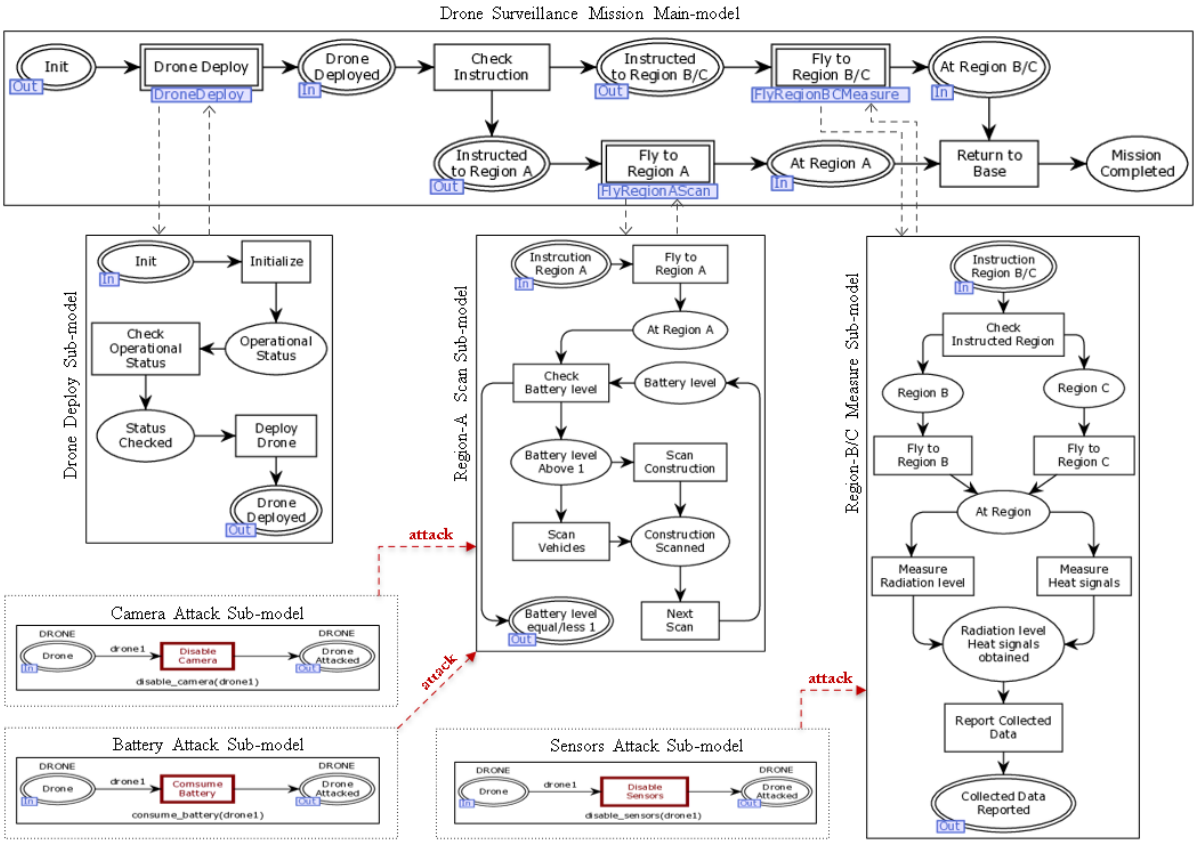

Figure 3: CPN Mission of the Surveillance Drone

Rule 2: Create input and output places for each transition, and create input and output arcs to pair the input place to the transition and the transition to the output place, respectively.

Rule 3: Produce a color set for each subject. The color set is a CPN datatype structure such as Product, Record, List, and Union [3]. The CPN color set combines primitive types such as *Bool*, *Int*, and *String*. It also can be a compound of primitive types and color sets. The color sets represent subject attributes. The color set that represents the Drone is generated as:

$color: Drone =$ **record** $\{location : String;$
$fly\_enabled : Bool;\ battery\_level : Int \in \{1,2,3,4\};$
$instruction\_issued : Bool;\ sensors\_enabled : Bool;$
$camera\_enabled : Bool;\ data\_collected : Bool\}$

Rule 4: Assign the color sets to the CPN places and arcs to declare their datatypes and declares a set of variables such that there is a variable for each color set. For instance, the declaration $var\ drone_1 : Drone$ declares a variable $drone1$ of type $Drone$.

Rule 5: Assign a guard expression for each CPN transition. The guard expression must be valid (evaluated true) in order for a transition to be executed (i.e., fired). For instance, $fly\_enabled\left(drone_1\right) = true$ is a guard expression assigned to the deployment transition. It ensures that the deployment transition fires only if the $drone_1.fly\_enabled$ attribute is assigned true.

Rule 6: Assign an arc expression for each input and output arc of a transition. The arc expressions evaluate tokens to pass in and out from transitions (i.e., bindings). When a token is evaluated true by an arc expression, it enables the transition that connects to be fired. For instance, $if\left(instruction\_issued\left(drone_1\right) = true\right)\{drone_1\} else\{empty\}$ is an arc expression that ensures the $drone_1$ token is instructed so the token passes; otherwise, the token does not bind the next transition, and the next transition will not be executed.

Rule 7: Initialize variables with values and assign these variables to the CPN places as tokens. The initialization forms the initial conditions of the mission. The tokens are then manipulated in each CPN transition, generating the state transitions of the CPN. For instance, the initial conditions of the surveillance drone mission are described as:
$\left(type\left(s\right) = Drone\right) \wedge \left(location\left(s\right) = \text{“}deployment\_point\text{”}\right) \wedge$
$\left(fly\_enabled\left(s\right) = true\right) \wedge \left(battery\_level\left(s\right) = 4\right) \wedge$
$\left(instruction\_issued\left(s\right) = false\right) \wedge \left(camera\_enabled\left(s\right) = true\right) \wedge$
$\left(sensors\_enabled\left(s\right) = true\right) \wedge \left(data\_collected = false\right)$

Figure 3 illustrates the *CPN* model resulting from applying rules 1-7 to the surveillance drone mission $\mathcal{M}_{drone}$. To maintain readability, we have explicitly omitted writing each arc expression into the *CPN* model. It shows the primary CPN mission, the Drone Surveillance Mission main model, and three sub-CPN models: Drone Deploy, Region A, and Region B/C sub-models. These three sub-CPNs are connected to the main CPN model through input and output ports, allowing the token to traverse through the sub-CPN model and back to the main CPN. The other three sub-CPNs, Camera Attack, Sensor Attack, and Battery Attack, present CPN attack models, which are discussed next.

5 THREAT MODELS AND ATTACK REPRESENTATION

In this section, we describe the threat model and attack representations for the surveillance drone mission.

5.1 Surveillance Drone Threat Model

The threat model is derived directly from the mission description (Step 1). The drone has mutable attributes that attackers can change. The other attributes, such as the drone type and location, are omitted as part of the threat model. For simplicity, we consider *battery_level*, *camera_enabled*, and *sensors_enabled* are mutable attributes. Table 1 illustrates the threat model for the surveillance drone mission.

Subject	Mutable Attribute	Attack Process	Impact on the Mission
Drone	*battery_level*	Consume one battery unit	It degrades the drone's ability to collect enough data and reduces flying time, leading to mission is *incomplete*; it might lead to mission *fails* if the drone does not have enough battery to fly back to the base
Drone	*camera_enabled*	Disable the camera	The drone is incapable of collecting data leading to mission is *incomplete*
Drone	*sensors_enabled*	Disable the sensors	The drone is incapable of collecting data leading to mission is *incomplete*

Table 1: Threat Model of the Surveillance Drone Mission

The attack degrading flying capability is considered to be consuming battery units. The battery consumption attack degrades the drone capability to collect much data since data collection requires battery. It also reduces the flying time of the drone (e.g., instead of flying 5 hours in a regular situation, it will fly 3 hours when an attack occurs). If the battery consumption attack occurs, we assume the drone should abort the mission and fly back to the base, and the mission is incomplete. Another possibility is that the drone does not abort, keeps flying, loses its battery, then is incapable of flying back to the base, and the mission fails.

If the camera or sensors are disabled by attacker, the drone cannot collect data but can fly back to the base, and the mission is incomplete. Notice that, besides mission *succeeds*, we mentioned two mission statuses: mission is *incomplete* and mission *fails*. Mission success indicates that the drone collects data and flies back to the base. The mission is incomplete indicates that the drone cannot collect data but can fly back to the base. Mission failure demonstrates that the drone cannot return to the base (e.g., lost). Analyzing the mission status from the mission specification is essential to translate the attacks into formal specification.

5.2 Formal Specification of Attacks

Each attack described in the threat model is formally specified as a CPN attack (i.e., a sub-CPN model). The CPN attack can be plugged into the proper place of the CPN mission based on precondition and postcondition. From the surveillance drone threat model in Table 1, three CPN attacks are formalized. These three CPN attacks are presented in Figure 3 as Camera Attack, Sensor Attack, and Battery Attack. These CPN attacks have the same precondition and postcondition as the Drone datatype. They must receive a variable of type Drone, change its values, and output a new variable of type Drone. Hence, these attacks can be attached to any place in the CPN mission if that place has a precondition and postcondition of Drone datatype. The CPN battery attack decreases the value of the drone variable's attribute $drone_1.battery\_level$ by 1 unit. The CPN Camera attack changes the value of the drone variable's attribute $drone_1.camera\_enabled$ to false, and the CPN sensor attack changes the value of the drone variable's attribute $drone_1.sensors\_enabled$ to false. For these changes in the drone's attributes values, resiliency

analysis is executed to verify where the CPN mission can succeed, incomplete, and failed.

6 RESILIENCY ANALYSIS

This section pursues applying the methodology (Steps 6 and 7) to the surveillance drone to analyze the drone mission and its resiliency. CPNTools [12] are used to analyze the state space and state transitions of the CPN mission to which CPN attacks are attached.

6.1 Drone Mission Specification Analysis

Various attack scenarios are created based on the drone threat model. An attack scenario consists of the subject of the attack, the set of attributes of the subject, which are attackable, and a limit on the number of times the attack may occur. A mission is resilient to an attack scenario if, after the attack occurs, there exists a successful execution from the point at which the attack occurred.

We investigate the decremental resiliency of a mission that expresses a mission-critical system is degraded during the execution of the mission, and no previously unavailable capability may recover. We start with one-shot resiliency, a version of decremental resiliency, referring to only one subject's capability being degraded. We then decrement the resilience with different attacks and degrade various capabilities to demonstrate a decremental resiliency analysis.

Resilience Type	Mission	Region	Attack	Status
One-shot	Scan buildings and construction sites	A	Disable camera	Incomplete
One-shot	Capture heat signals and radiation levels	B	Disable sensors	Incomplete
One-shot	Capture heat signals and radiation levels	B	Disable camera	Succeed
Decremental	Scan buildings and construction sites	A	Consume battery many times	Fail
Decremental	Scan buildings and construction sites	A	Consume battery once and disable camera	Incomplete

Table 2: Verification Attack Scenarios

Table 2 examines five attack scenarios where the drone is at regions A or B. The first three attack scenarios investigate one-shot resiliency where one attribute (camera, battery, or heating and radiation sensors) is disabled. The other two attack scenarios investigate decremental resiliency where the battery and camera are consumed and disabled, respectively. The first attack scenario expresses that the drone's camera fails while scanning region A. The drone had survived and returned safely to the base, but the mission is incomplete because it could not collect data from the region. The second attack scenario shows the drone's sensors are disabled while measuring the heat signals and radiation levels at region B. The drone returned safely to the base, but the mission is incomplete since data collection failed. The third attack scenario disables the camera at Region B; however, the mission succeeds since the drone does not use the camera but uses sensors to collect data and can safely return to the base. The fourth attack scenario consumes the battery repeatedly until the drone loses the power to return to the base. We refer to this situation as a mission failure. The fifth attack scenario consumes the battery one time and disables the camera. The drone can return to the base, but the mission is incomplete because it cannot collect data.

6.1.1 State Space Analysis. The analysis examines the state space of the CPN model with each attack scenario. Table 3 reports that all CPNs of attack scenarios are strongly connected components (SCC). It indicates that the state models generated from these CPNs are acyclic; they do not have infinite loops. However, it reports dead markings and dead transitions for each attack scenario, which are our considerations for verification. Dead markings are states where the CPN is terminated, and dead transitions are when the transitions are not executed (no token visits them). Verifying the dead markings signifies the values of the drone' attributes where the mission is terminated, whereas verifying the dead transitions signifies why the drone did not execute specific tasks.

Region & Attack Scenario	State Space		SCC Graph		Status
Region-A, Disable Camera	#Node 19	#Arcs 21	#Node 19	#Arcs 21	Full
	Dead Markings [19]		#Dead Transition 8		Live Transition None
Region-B, Disable Sensors	#Node 12	#Arcs 12	#Node 12	#Arcs 12	Full
	Dead Markings [12]		#Dead Transition 9		Live Transition None
Region-B, Disable Camera	#Node 12	#Arcs 12	#Node 12	#Arcs 12	Full
	Dead Markings [12]		#Dead Transition 10		Live Transition None
Region-A, Consume Battery Many Times	#Node 15	#Arcs 16	#Node 15	#Arcs 16	Full
	Dead Markings [15]		#Dead Transition 9		Live Transition None
Region-A, Consume Battery One Time and Disable Camera	#Node 18	#Arcs 19	#Node 18	#Arcs 19	Full
	Dead Markings [18]		#Dead Transition 7		Live Transition None

Table 3: State Space Report

As mentioned, the first three attack scenarios examine one-shot resiliency. The state space of the first attack scenario, Region-A & Disable Camera, reports one dead marking, 19, and 8 dead transitions. The second attack scenario, Region-B & Disable Sensors, shows one dead marking, 12, and 9 dead transitions. Similarly, for the third attack scenario, Region-B & Disable Camera, it reports one dead marking, 12, but 10 dead transitions. Even though the second and third attack scenarios are similar, the second attack scenario was incomplete with 9 dead transitions, and the third attack scenario succeeds with 10 dead transitions. The 10th dead transition of the third scenario is considerably firstly to be verified.

The fourth and fifth attack scenarios examine decremental resiliency. The state space of the fourth attack scenario, Region-A & Consumes Battery Many Times, reports dead marking, 15, and 9 dead transitions. The fifth attack scenario, Region-A & Consume Battery One Time and Disable Camera, reports dead marking, 18, and 7 dead transitions. Next, the dead markings and dead transitions are verified and explained.

6.1.2 Formal Verification. We use CPN-ML programming language [8] to verify dead marking and dead transition for each attack scenario. We also use built-in functions provided by the CPNTools [12], such as *Reachable(x,y)*, *SccReachable(x,y)*, *AllReachable()*, *DeadMarking(x)*, *ListDeadMarkings()*, and *ListDeadTransitions()*, for the verification process. These functions return an execution sequence for each dead marking where the mission is succeeded, incomplete, or failed. We backtrack through the execution sequence and locate the state where the attack occurs. While backtracking, we verify the change of states in the sequence to inspect the reasons that lead to dead transitions and what state values the sequence terminates with at dead markings.

Verification of the state space of the first scenario (One-shot at Region-A, Disable Camera) shows that the drone was instructed at State 4 to fly to Region A to scan buildings and construction sites. State 6 shows the drone was in region A. The attack occurred at the State 8, and the camera was disabled. States 9 to 18 show that the drone flew over Region A three times, consuming the battery and leaving only one battery unit, and could not collect data. In State 19, the CPN terminated with the drone returned to the base and mission status "Incomplete" because no data had been collected.

In the second scenario (One-shot at Region B, Disable Sensors), the drone was instructed in State 4 to fly to Region B to measure heat signals and radiation levels, and in State 6, the drone was in Region B. The attack occurred at State 8, and sensors were disabled. In States 9,10, and 11, the drone consumed one battery unit and could not collect data. In State 12, the CPN terminated with the drone returned to the base and mission status "Incomplete".

The third scenario (One-shot at Region B, Disable Camera) is similar to the second scenario; the drone instructed at State 4 was in Region B at State 6, and the attack occurred at State 8. The difference is that the camera was disabled, but not the sensors. States 9, 10, and 11 show that the drone was able to collect data, and in State 12, the CPN terminated with the drone returned to the base and mission status "Succeed". Verifying the 10th dead transition of the third scenario that differs from the second scenario, we found that the dead transition is "SensorsAttack'Disable_Sensors 1" and did not execute because the attack disabled the camera but not the sensors, which caused the drone to be able to collect data, return to the base, and the mission succeeded.

In the fourth scenario (Decremental at Region-A, Consumes battery Many Times), the drone instructed at State 4 was in Region A at State 6, the attack occurred at State 7, and the battery consumed one unit, leaving it with 3 units remaining. In State 10, the drone continuously flew over Region A, using the camera to scan the region and consuming one battery unit, leaving it with 2 units remaining. In State 11, another attack occurred, consuming one more battery unit, leaving it with 1 unit. State 14 shows the drone used this 1 battery unit for scanning, leaving it with zero unit. In State 15, the CPN terminated with the drone did not return to the base and the mission status "Fail".

For the last decremental scenario (Decremental at Region-A, Consumes battery One Time and Disable Camera), in State 7, the attack occurred and disabled the drone's camera first, and then another attack occurred in State 10, consuming one battery unit, leaving it with 3 units. From States 11 to 16, the drone continuously flew over region A, consuming its battery, but was unable to use the camera. In State 17, the battery reached one unit remaining, and the drone decided to return to the base. The CPN terminated in State 18, showing that the drone returned to the base with the mission status "Incomplete".

6.2 Drone Resiliency Analysis

The verification indicates that the drone must immediately abort the mission when data collection is infeasible. Continuous flying and consuming the battery without collecting data minimizes the resilience level of the mission. Table 4 expresses constraints that enhance the resiliency of the surveillance drone mission.

Resilience Type	Mission	Region	Mission Restriction
One-shot	Scan buildings and construction sites	A	Abort the mission if the camera is disabled
One-shot	Capture heat signals and radiation levels	B	Abort the mission if the sensors is disabled
One-shot	Capture heat signals and radiation levels	B	Abort the mission if the task is infeasible
Decremental	Scan buildings and construction sites	A	Save enough battery to fly back to the base
Decremental	Scan buildings and construction sites	A	Save battery and abort the mission if the task is infeasible

Table 4: Mission Restrictions

In summary, we have shown how to assess a mission's decremental resiliency and find the conditions required for the mission to succeed. We can be confident that a mission can succeed under the restricted workflow even with the occurrence of tasks' degradation.

7 RELATED WORK

The literature on workflow resiliency problems introduces solutions to address the unavailability [9, 10, 14, 15]. Our work argues that the workflow resiliency problem can be viewed as degradation.

Wang *et al.* [14] introduce three types of resilience, static, decremental, and dynamic. They use the term a user to refer to a subject, an active entity doing tasks. Static resilience refers to a situation in which subjects become unavailable before the workflow executes, and no subjects may become available during the execution. Decremental resiliency expresses a condition where subjects become unavailable before or during the execution of the workflow, and no previously unavailable subjects may become available during execution, while dynamic resilience describes the situation where a subject may become unavailable at any time; a previously unavailable subject may become available at any time. The different types of resilience formulations capture various types of attack scenarios.

Yang *et al.* [15] address the workflow satisfiability problem that ascertains whether a set of subjects can complete a workflow. The authors investigate the computational complexity of the workflow satisfiability problem in two aspects. One aspect considers either one path or all paths of a workflow, and the other focuses on the possible patterns in a workflow. They present a set of algorithms for solving various types of problems of workflow satisfiability. They show that many existential and universal workflow satisfiability problems are NP-complete and NP-hard. Thus, they conclude that restrictions on

workflow patterns induce such problems to be solvable in polynomial time.

Mace *et al.* [9, 10] propose a quantitative measure of workflow resiliency. They use a Markov Decision Process (MDP) to model workflow to provide a quantitative measure of resilience. They refer to binary classification, such as returning an execution sequence if one exists and declaring the workflow resilient or returning false and declaring the workflow not resilient. The authors show that the MDP models give a termination rate and an expected termination step.

Abdelgawad *et al.* [1] address the static resiliency of mission mission-critical system. The authors use workflow to describe a mission for a mission-critical system. They then formalize rules that convert the mission workflow into CPN. The formalization covers a mission workflow that includes many subjects and objects. The work focuses on verifying the static resiliency of the mission workflow.

Our methodology leverages other works for analyzing workflow, using formal specifications, and investigating resiliency. We design the methodology to cover all aspects of mission resiliency, static, decremental, and dynamic resiliency. We differ from others in a systematic way of analyzing mission specifications, constructing mission workflow, generating threat models, and investigating various attack scenarios that exercise mission resiliency. This methodology provides reasonable restrictions that increase the resiliency of missions for mission-critical systems.

8 CONCLUSION

This paper introduces a methodology that addresses the mission resiliency analysis for mission-critical systems. The methodology systematically analyzes various aspects of mission resiliency: static, incremental, and dynamic. It consists of seven steps, starting from a description of a mission, a formal representation of the mission, converting the mission's formal representation to formal specification, constructing threat models, formal specification of attacks, and ending with resiliency analysis. The methodology uses workflow for the formal representation of the mission. It also utilizes Coloured Petri Nets (CPN) to formally specify the mission and attacks.

We applied the methodology to a drone surveillance system as a motivating example. CPN tools are used to verify various attack scenarios. The results show that mission resiliency is a degradation issue. When an attack occurs, a mission-critical system can continue, and the mission can be accomplished. The methodology is practical to explore the state transitions of a mission and verify which state the mission succeeds, fails, and is incomplete. It provides restrictions that must be added to the mission description to improve resiliency.

Future work will investigate the applicability of the methodology to various mission-critical system application domains. It will also investigate the effectiveness and scalability of the methodology on complicated mission-critical systems that incorporate many subjects, workflows, and objectives. These future investigations will unveil the challenges of time and memory used to verify large tasks.

ACKNOWLEDGMENTS

This work was supported in part by funding from NSF under Award Numbers DMS 2123761, CNS 1822118, NIST, ARL, Statnett, AMI, NewPush, and Cyber Risk Research.

REFERENCES

[1] Mahmoud Abdelgawad, Indrakshi Ray, and Tomas Vasquez. 2023. Workflow Resilience For Mission Critical Systems. In *Proceedings of the 25th International Symposium of Distributed Systems Stabilization, Safety, and Security SSS*, Vol. 14310. Springer, Jersey City, NJ, USA, 498–512.

[2] Jennifer Chong, Partha Pal, Michael Atigetchi, Paul Rubel, and Franklin Webber. 2005. Survivability architecture of a mission critical system: The DPASA example. In *The proceeding of the 21st Annual Computer Security Applications Conference (ACSAC)*. IEEE Computer Society, Tucson, AZ, USA, 495–504.

[3] CPN Tools Administration. 2018. Help topics for verification. http://cpntools.org/2018/01/15/verification/.

[4] Philip W. L. Fong. 2019. Results in Workflow Resiliency: Complexity, New Formulation, and ASP Encoding. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 185–196.

[5] Kyriakos Houliotis, Panagiotis Oikonomidis, Periklis Charchalakis, and Elias Stipidis. 2018. Mission-critical systems design framework. *Advances in Science, Technology and Engineering Systems Journal* 3, 2 (2018), 128–137.

[6] Kurt Jensen, Lars Kristensen, and Lisa Wells. 2007. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *STTT* 9 (05 2007), 213–254.

[7] Kurt Jensen and Lars Michael Kristensen. 2009. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer International Publishing, Cham.

[8] Kurt Jensen and Lars M. Kristensen. 2009. *CPN ML Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 43–77.

[9] John C. Mace, Charles Morisset, and Aad van Moorsel. 2015. Impact of Policy Design on Workflow Resiliency Computation Time. In *Proceedings of the Quantitative Evaluation of Systems (QEST)*, Vol. 9259. Springer, Madrid, Spain, 244–259.

[10] John C. Mace, Charles Morisset, and Aad van Moorsel. 2014. Quantitative Workflow Resiliency. In *Proceedings of the Computer Security (ESORICS)*, Mirosław Kutyłowski and Jaideep Vaidya (Eds.), Vol. 8712. Springer, Wroclaw, Poland, 344–361.

[11] Christophe Ponsard, Philippe Massonet, Jean-François Molderez, André Rifaut, A van Lamsweerde, and H Tran Van. 2007. Early verification and validation of mission critical systems. *Formal Methods in System Design* 30, 3 (2007), 233–247.

[12] Anne V. Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. 2003. CPN Tools for Editing, Simulating and Analysing Coloured Petri Nets. In *The proceeding of the Applications and Theory of Petri Nets 2003: 24th International Conference (ICATPN)*, Wil M. P. van der Aalst and Eike Best (Eds.), Vol. 2679. Springer, Eindhoven, The Netherlands, 450–462.

[13] Ron Ross, V Pillitteri, R Graubart, D Bodeau, and R McQuaid. 2019. NIST Special Publication 800-160 Volume 2: Developing Cyber Resilient Systems. *NIST Special Publication* 2 (2019), 224.

[14] Qihua Wang and Ninghui Li. 2010. Satisfiability and Resiliency in Workflow Authorization Systems. *ACM transactions on information and system security* 13, 4 (2010), 1–35.

[15] Ping Yang, Xing Xie, Indrakshi Ray, and Shiyong Lu. 2014. Satisfiability Analysis of Workflows with Control-Flow Patterns and Authorization Constraints. *IEEE Transactions on Services Computing* 7, 2 (2014), 237–251.