# Generating Cyber-Physical System Risk Overlays for Attack and Fault Trees using Systems Theory

Matthew Jablonski, Duminda Wijesekera
George Mason University
Fairfax, VA, USA
mjablons@gmu.edu,dwijesek@gmu.edu

Anoop Singhal
National Institute of Standards and Technology
Gaithersburg, MD, USA
anoop.singhal@nist.gov

## ABSTRACT

We describe a formalized systems theoretic method for creating cyber-physical system (CPS) risk overlays that augment existing tree-based models used in CPS risk and threat analysis processes. This top-down approach objectively scopes the system's threat surface for some risk scenario consequence by analyzing its underlying control attributes and communication flows between relevant internal hardware and software sub-components. The resulting analysis should assist with the qualitative selection of causal events when utilizing attack and fault tree models, which have traditionally conducted this event selection using subjective and bottom-up methods. Objectively scoping the tree-based model analysis using a proven systems theoretic approach should also improve defensive and safety planning during the system development life cycle. We provide a control system case study using attack-defense trees and show how this approach may also be reduced to attack trees, fault trees, and attack-fault trees.

**ACM Reference Format:**
Matthew Jablonski, Duminda Wijesekera and Anoop Singhal. 2022. Generating Cyber-Physical System Risk Overlays for Attack and Fault Trees using Systems Theory. In *Proceedings of the 2022 ACM Workshop on Security and Trustworthy Cyber-Physical Systems (SaT-CPS '22), April 27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3510547.3517922

An incident occurring in a cyber-physical system (CPS) may be due to a fault, a complex series of logical or physical failures, or an intentional disruption or control alteration that affects the safety ensured by the physical components. Safety and security experts manage these risk scenarios by analyzing the possible chains of events that lead to such failures and constraining the system's behaviors through standard-based design and other best practices.

Tree-based risk models, such as fault trees [16], attack trees [11, 12], and attack-defense trees [1, 3, 4, 6, 7], are modeling tools used to qualitatively and quantitatively assess CPS risks. These tools take input from the system's architecture along with data from safety and security standards and frameworks to produce an output graph of possible issues that may arise for further analysis. The benefits from the analyzed output are due to the simplistic relational view of possible issues that experts may use in risk management planning. They also have a variety of well-researched methods for quantitative analysis of risk.

A general problem with these models recognized in systems theory is that although they are useful in relating identified events that realize some risk, they provide little guidance for event selection [9]. They do not provide a comprehensive approach in identifying paths that some threat actor(s) may take to realize a given consequence within a complex system. As such, event selection by analysts is subjective and prone to human error or bias due to their limited experience.

In practice, CPS issues arise due to the related complexities of having multiple control systems synchronized at different levels of granularity within the overall system. As these systems control networked physical equipment, different events may result in similar impacts to the functioning state of the system and/or other physical effects. Such a state change may be considered intentional (through attacks) or accidental (through faults) which are respectively handled as security or safety risks or some combination. In addition, humans-in-the-loop involvements may create additional paths in terms of errors and delays. Tree-based risk models may be used to identify and analyze these complex event relationships if an objective approach is first taken to properly scope their analysis.

We address these shortcomings in tree-based risk models by adapting a system theoretic approach to frame the search space within a system for event selection. To meet this objective, our approach may be described in three steps: (1) Utilize Systems Theoretic Process Analysis (STPA [15]) to identify the operational space of undesirable activity of a CPS system by combining attack surfaces with the space of safety- and reliability-related faults, which we term a *threat surface.* (2) Identify *Potential Hazardous Control Action* (PHCA) attributes to select tactically relevant events. (3) Use a top-down construction method to convert the threat surface to paths within a tree-based risk model, which we term a *CPS risk overlay.*

A high-level visualization of our approach is shown in Figure 1. Here, we take some generic CPS, analyze its threat surface through a systems theoretic approach, and generate a CPS risk overlay, which may be further analyzed through a tree-based model to identify and address faults and attacks. In this work, we formalize our approach and demonstrate the application of a risk overlay towards attack-defense trees. We provide a reduction method that applies the attack-defense tree modifications towards other tree-based models, specifically discussing attack trees, fault trees, and attack-fault trees. We analyze an elevator control system as a case study.

*SaT-CPS '22, April 27, 2022, Baltimore, MD, USA.*
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9229-7/22/04.
https://doi.org/10.1145/3510547.3517922

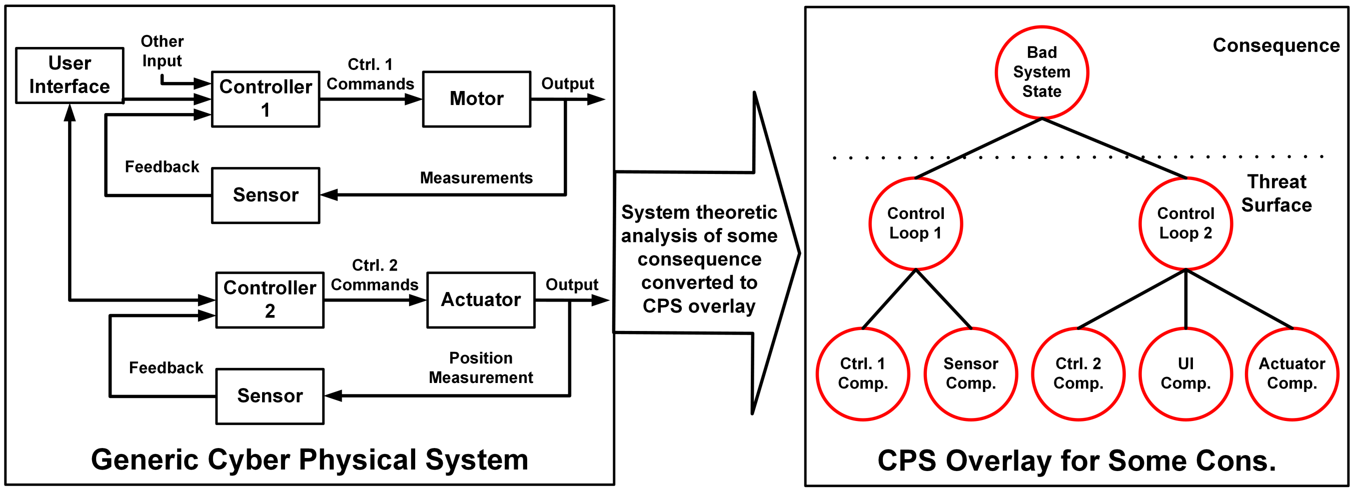


**Figure 1: Converting a CPS threat surface for some system-relevant consequence to a CPS risk overlay.**

The rest of the paper is written as follows. Section 1 presents related work. Section 2 describes the construction of our risk overlay, and Section 3 describes how attack-defense trees (ADTrees) may be augmented to improve analysis when using risk overlays. Section 4 shows our case study of applying the risk overlay method. Section 5 concludes the paper.

## 1 RELATED WORK

Leveson et al.'s Systems Theoretic Process Analysis (STPA [10]) is an accident causality model that uses a systems theoretic approach to identify threats to control systems and is based on their work on Systems-Theoretic Accident Model and Processes (STAMP [9]). Originally intended for safety risk analysis, STPA was extended to security risk analysis in STPA-Sec [17]. We utilize a formalized STPA process, derived from [15], for identifying the threat surface of a particular consequence and then construct a CPS risk overlay for existing tree-based models. We formalize this method to describe a repeatable and objective that may be adapted towards tree-based risk models. Other methods, such as Consequence-Driven, Cyber-Informed Engineering (CCE [2]), take a similar systems theoretic top-down approach in system analysis for security research, and we see our work as an augmentation that assist with the integration of tree-based risk models within such approaches.

We leverage Kordy et al.'s ADTree model [7] as an example tree-based model that facilitates the inclusion of defensive actions within their qualitative and quantitative analysis. By assigning attribute values to basic attacker and defense actions, ADTrees facilitate the quantification of risk scenarios using bottom-up calculations [1]. Additional ADTree semantics and extensions introduced in literature include propositional semantics, those induced by De Morgan lattices, multiset and equational semantics [7], semantic extensions handling repetitions of node labels using twin and cloned nodes [3], semantic extensions incorporating sequential conjunctions [4], and the equivalence of ADTrees with two-player binary zero-sum extensive form games [6]. To the best of our knowledge, no current work applies systems theory and ADTree attributes to safety and security risk analysis in CPSs, which we show through our augmented model called CPS-ADTrees.

## 2 CONSTRUCTING A CPS RISK OVERLAY

We now describe the CPS risk overlay construction process for a given risk scenario within a CPS. In the rest of paper uses the general vector notations for an $i$-dimensional vector at time step $t$, such that $\overrightarrow{vec}_{i,t} =< vec_{0,t}, \ldots, vec_{i-1,t} >$.

### 2.1 Consequence Identification

The events that we want to prevent are those that impact its business or mission objectives [9], where each identified consequence may be the top node in a tree-based model, represented as $Cons = \overrightarrow{out}_i^{NCAct}$. The vectors $\overrightarrow{out}_i^{NCAct}$ are the top node's attributes representing $i$ notional output control actions that constitute the risk scenario. We use these attributes to identify the related threat surface.

### 2.2 Threat Surface Identification

We analyze the consequence of the chosen scenario by looking for events that can be triggered by the system if the identified scenario were to occur. This search identifies the threat surface by analyzing the data handling processes within system components and communication interfaces relative to $\overrightarrow{out}_i^{NCAct}$. The result of this analysis is a subset of control loops, along with their internal components and their attributes that are further analyzed and added hierarchically as nodes of the CPS risk overlay.

*2.2.1 Control Loop Components.* We begin by defining component and the attributes that change the state of a CPS. We model the behavior of the $k^{th}$ component at the $t^{th}$ time step as an input-output pair, $ioComp_{k,t} = (\overrightarrow{in}_{k,a,t}, \overrightarrow{out}_{k,b,t})$. The vectors $\overrightarrow{in}_{k,a,t}$ and $\overrightarrow{out}_{k,b,t}$ represent $a$ input and $b$ output signals as attributes associated with component $k$ at time step $t$. These input and output attributes constitute a part of the threat surface.

### 2.2.2 *Communication Interfaces.*

The communication paths between the $m^{th}$ and $n^{th}$ component, for some $m, n \in \mathbb{N}$, are formalized using the (directional) binary predicate *connected(Src,Des)*, as $\forall i \le x, \exists j \le y : connected(out_{m,i,t}, in_{n,j,t})$.

This path definition implies that the connectivity among components are transitively closed under information and control flows. Identified paths also become a part of the threat surface. If $m = n$, then the communication path may represent some internal process communication included within the component threat surface. Otherwise, components along paths (such as routers, hubs, or switches) should be considered as another type of $ioComp_{k,t}$ and as part of the threat surface.

### 2.2.3 *Controller Process Model.*

A controller is an $ioComp_{k,t}$, that converts its inputs to outputs to manage the control loop. Irrespective of a human or digital controller, we model the $k^{th}$ controller executing its $t^{th}$ time step as a six-tuple, where for some $a, b, c, d \in \mathbb{N}$, we have $ctrl_{k,t} = (\overrightarrow{in}_{k,a,t}, \overrightarrow{out}_{k,b,t}, \overrightarrow{pMod}_{k,c,t}, \overrightarrow{ioComp}_{k,d,t}, cObj_{k,t}, \chi)$. Here, $\overrightarrow{in}_{k,a,t}$ and $\overrightarrow{out}_{k,b,t}$ respectively represent $a$ input and $b$ output values of the controller. The vector $\overrightarrow{pMod}_{k,c,t}$ contains the $c$ internal process model values of the $k^{th}$ controller that describe its context of the environment, which may be updated. The vectors $\overrightarrow{ioComp}_{k,d,t}$ are the $d$ components within the $k^{th}$ controller's control loop, that provide input data or actuate the physical process. All of these attribute vectors also constitute the controller's threat surface.

The remaining definitions model how a controller makes the decision to change the physical system state. The local control objective at time step $t$ is $cObj_{k,t}$, which may be dynamically derived from some global control objective. The input-output behavior of the control loop during the $t^{th}$ time step is modeled as the response function $\chi$, where $\chi(\overrightarrow{pMod}_{k,c,t}, cObj_{k,t}) = \overrightarrow{out}_{k,b,t}$. The function $\chi$ abstracts traditional control functions such as those found in a proportional–integral–derivative (PID) controller or in model predictive control (MPC) [14].

## 2.3 Potential HCAs

Now we have the threat surface identified as a collection of attribute vectors that could be used by a threat actor to drive the system state towards the consequence. Any action in this list is a Potential Hazardous Control Action (PHCA), referred to as a Hazardous Control Action (HCA) in [15]. Our objective here is to determine subsets of actions that could lead to the undesirable consequence.

We model the $a^{th}$ PHCA for controller, $ctrl_{k,t}$, at time $t$ as a four-tuple, $PhazCA_{k,a,t} = (\overrightarrow{pMod}_{k,b,t}, cObj_{k,t}, \overrightarrow{out}_{k,c,t}, timeChk(t))$, for some $b, c \in \mathbb{N}$. Possible values for inputs and outputs are mapped back to the control function $\chi$, and a PHCA may result if this response is not provided in a timely manner, represented by the function $timeChk(t)$, described as follows. Let $t_{ex}$ and $t_{act}$ represent time steps when $\overrightarrow{out}_{k,d,t}$ is expected to be received versus when it is actually received at the actuator. When $timeChk(t) = 1$, these moments are approximate enough to provide the response in a timely manner, such that $t = t_{act}$ and $t_{ex} \approx t_{act}$. Similarly, $timeChk(t) = 0$ if the signal is not provided at the expected time. The PHCA model is a table entry mapping inputs and the control objective to outputs along with $timeChk(t)$.

To summarize the utility of $timeChk(t)$, a PHCA may result either when $\overrightarrow{out}_{k,c,t}$ is an improper response when provided in context $\overrightarrow{pMod}_{k,b,t}$ with control objective $cObj_{k,t}$ at time $t$, represented by $timeChk(t) = 1$. Or it could result when $\overrightarrow{out}_{k,c,t}$ is not provided at the proper time given the same inputs, where $timeChk(t) = 0$.

### 2.3.1 *Control Loop Analysis.*

PHCAs within a controller are identified by analyzing the components within their control loop. We represent this control loop consequence analysis $cLCons_k$ for analyzing the consequence, $Cons$, within controller $ctrl_{k,t}$ as a four-tuple where $cLCons_k = (Cons, hChk(), hNChk(), cLChk())$. The value returned by the hazard check function $hChk(Cons, PhazCA_{k,a,t})$ is equal to 1 if and only if $Cons$ results from the $k^{th}$ controller executing $\overrightarrow{out}_{k,c,t}$ to satisfy $\chi$ at time $t$. The value returned by the untimely hazard check function $hNChk(Cons, PhazCA_{k,a,t})$ is equal to 1 if and only if $Cons$ results from the $k^{th}$ controller not executing $\overrightarrow{out}_{k,c,t}$ to satisfy $\chi$ at time $t$. The value returned by the control loop check function $cLChk(PhazCA_{k,a,t})$ is equal to 1 if and only if the $k^{th}$ controller is required to execute $\overrightarrow{out}_{k,c,t}$ to satisfy $\chi$ within the control loop at time $t$. This analysis may result in 3 different scenarios for a PHCA:

(1) A PHCA is an HCA for $Cons$ if its execution results in $hChk() = 1$ and $cLChk() = 0$, which occurs when the action represented by the PHCA results in $Cons$ within an execution of the control loop regardless the value of $timeChk(t)$.
(2) A PHCA may be an HCA if the execution results in $hNChk() = 1$ when $timeChk(t) = 0$, indicating that proper timing is required for this action to be non-hazardous. Otherwise, if the PHCA is executed at the proper time when $timeChk(t) = 1$, then $cLChk() = 1$, and it may not be an HCA.
(3) Any PHCA where $hChk() = 0$ and $hNChk() = 0$ is not considered hazardous and results in $cLChk() = 1$. These control actions are not HCAs for $Cons$.

At the system level, we can use these checks to assure that $Cons$ does not occur when all $k$ controllers execute $\overrightarrow{out}_{k,c,t}$ by satisfying their response functions $\chi$ at time $t$. This assurance is due to limiting each controller only to the actions resulting in $cLChk() = 1$. The analysis of PHCAs at the control loop and system level is the reason we transform the hierarchical control loop and component structure to a CPS risk overlay so that we may further identify and assess events leading up to an HCA using a tree-based model.

## 2.4 CPS Risk Overlay Construction

Using the attributes identified for the consequence and threat surfaces, we now construct the CPS risk overlay using the top-down method in Algorithm 1, Construct_CPS_Risk_Overlay. The resulting CPS risk overlay qualitatively groups components by their connected control loop communication structure.

Inputs to the algorithm include the analyzed consequence, $Cons$, and the threat surface comprised of $k$ controllers, $\overrightarrow{ctrl}_k$, as described in lines 1 and 2. Line 3 describes the output as the constructed CPS risk overlay, $topNode$. Line 4 sets $topNode$ to the input consequence node. Line 5 initiates a for-loop with counter $i$ that steps

**Algorithm 1** : Construct_CPS_Risk_Overlay

```
1:  //input: Cons = consequence for risk scenario
2:  //input: $\overrightarrow{ctrl}_k$ = vector of $k$ controllers in system
3:  //output: topNode = representative tree root for Cons
4:  topNode ⟵ Cons
5:  for $i = 0$ to $k - 1$ do //Assume $k$ controllers
6:      topNode.add_OR($ctrl_i$ CL)
7:      cLNode ⟵ topNode.get_child($ctrl_i$ CL)
8:      cLNode.add_OR($ctrl_i$)
9:      for $j = 0$ to $l - 1$ do //Assume $\overrightarrow{ioComp}_{i,l}$
10:         //Add check for PHCA relation
11:         cLNode.add_OR($ioComp_{i,j}$)
12:         //Add loop for comm paths, if desired
13:     end for
14: end for
```

through each controller. Lines 6 and 7 adds the control loop to the tree, named "$ctrl_i$ CL", as a disjunctive child to the top node. Line 8 adds the controller $ctrl$ to the tree as a disjunctive child node to the control loop. Line 9 through 13 initiates a for-loop with counter $j$ that steps through all $l$ components that communicate with the controller and adds each as a disjunctive child to the control loop. If some check is added to determine if a component affects the PHCAs associated with $Cons$, then line 10 notes we could add further procedures to conduct this check. If a communications path between some components is identified with the transitive closure property, then line 12 notes that we could add further procedures to loop through each intermediary communication node and add them as threat surfaces. When the for-loop invoked in line 5 completes in line 14, $topNode$ represents the CPS risk overlay for consequence $Cons$.

## 3 CPS ATTACK-DEFENSE TREES

We now describe how to apply the CPS risk overlay to an ADTree using node attributes as well as other tree-based models through reduction.

### 3.1 Attack-Defense Tree Review

An ADTree is a tree consisting of nodes of partial instantiations of attributes with a label (AND, OR, SAND = sequential AND), name, and a type (either proponent or opponent). The type of the root node is referred to as the model's proponent and models a desirable or an undesirable scenario regarding a state of the system. The root node may be either an attack node or a defense node. The opposite type represents the opponent. A node in the ADTree can have one child with the opposite type (representing a countermeasure) or more than one child of the same type and a label. The children of nodes labeled as AND, OR, or SAND represent refinements of their parent. A node without refinements is called a basic action. Attack nodes are typically shown as red circles and defense nodes as green squares. The right hand side of Figure 1 shows an ADTree with only attack nodes. The intermediate nodes are OR nodes refined with children. Our case study show a more detailed ADTree in Figure 3.

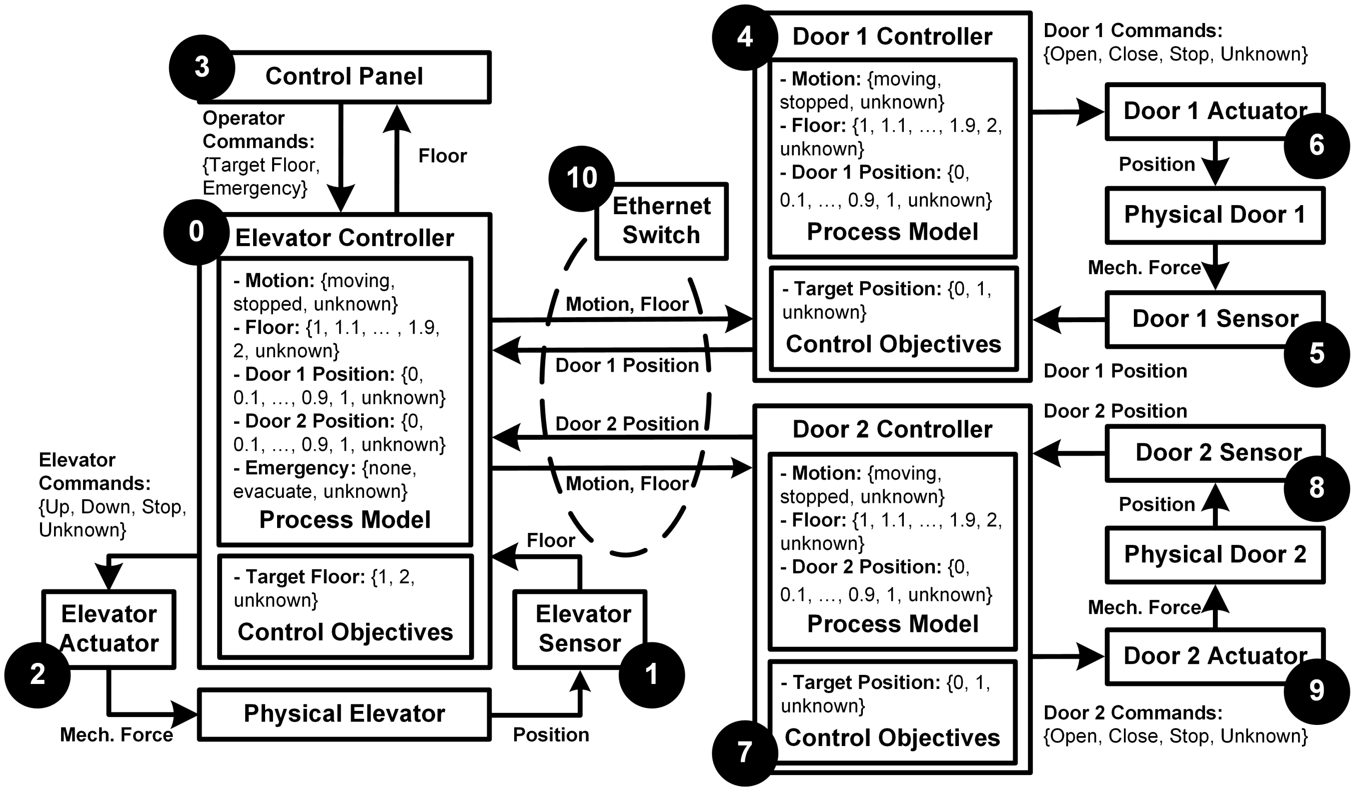
### 3.2 Overlaying CPS Attributes and Reduction to Other Risk Models

A CPS-ADTree is an ADTree with additional attributes to model both attacks and faults and their related countermeasures. To differentiate between unintentional and attack-caused events within CPS-ADTree models, we apply attributes to basic actions to denote them as being safety- or security-related. With this attribute modifications, CPS-ADTrees represent a two-person game focused on causing or preventing some system state as defined by the risk scenario. Our top-down construction method resulted in a CPS risk overlay that may be applied to the CPS-ADTree. An example of this risk overlay can be seen by the shaded threat nodes identified in the case study discussed in Section 4. The paths through the risk overlay must be traversed by the threat to cause the consequence. Each node may be further pruned, refined, or adjusted per ADTree semantics. These refinements should derive and define possible threat paths and defensive countermeasures, applying known threat modeling techniques, such as those found in MITRE ATT&CK® [13], which identify known threats by tactics, techniques, and procedures. These refinements allow for analysis to plan realistic defensive strategies, if desired.

A CPS-ADTree may be reduced to a fault tree [16] / attack tree [11, 12] if we prune subtrees consisting of only basic actions with safety / security attributes along with all subtrees rooted in defense nodes. Similarly, a CPS-ADTree may be reduced to an attack-fault tree [8] by only pruning subtrees rooted in defense nodes. A reduced tree represents all paths that leads to realizing the risk scenario despite installed defensive measures. By using these reductions one could apply use our CPS risk overlay to other well-researched tree-based models. This reduction allows for structuring trees according to their definitions and would require no additional changes to existing modeling tools to support the risk overlay method, as the assessor would simply need to add the intermediate control loop and sub-component nodes within these tools when framing the tree.

## 4 CASE STUDY

We consider an elevator system situated in a high-traffic building as a case study. Our example elevator services two floors and may move up or down to the opposite floor based on the elevator's current position and the operator input to a control panel. Each floor has a door that opens and closes for passengers, and the elevator should only move to the opposite floor when both doors are closed. It does not change direction during the upward or downward journey, but it may stop for an emergency. The operator may only press a button for the target floor and/or push a separate emergency button to stop movement. The doors must only open once the elevator stops moving and rests in the appropriate vertical position. The doors may be opened manually if the elevator has performed an emergency stop between floors. After some time, the doors will close allowing the elevator to move to the opposite floor if some user input is provided. Connections between the operator panel, the elevator controller, and a separate controller for each door are networked using the Modbus protocol over TCP/IP through wired Ethernet connections on a standalone local area network (LAN). Our



Control Panel
Operator Commands: {Target Floor, Emergency}
Floor
Elevator Controller
- Motion: {moving, stopped, unknown}
- Floor: {1, 1.1, … , 1.9, 2, unknown}
- Door 1 Position: {0, 0.1, …, 0.9, 1, unknown}
- Door 2 Position: {0, 0.1, …, 0.9, 1, unknown}
- Emergency: {none, evacuate, unknown}
Process Model
- Target Floor: {1, 2, unknown}
Control Objectives
Elevator Commands: {Up, Down, Stop, Unknown}
Elevator Actuator
Mech. Force
Physical Elevator
Position
Elevator Sensor
Floor
Ethernet Switch
Motion, Floor
Door 1 Position
Door 2 Position
Motion, Floor
Door 1 Controller
- Motion: {moving, stopped, unknown}
- Floor: {1, 1.1, …, 1.9, 2, unknown}
- Door 1 Position: {0, 0.1, …, 0.9, 1, unknown}
Process Model
- Target Position: {0, 1, unknown}
Control Objectives
Door 1 Commands: {Open, Close, Stop, Unknown}
Door 1 Actuator
Position
Physical Door 1
Mech. Force
Door 1 Sensor
Door 1 Position
Door 2 Controller
- Motion: {moving, stopped, unknown}
- Floor: {1, 1.1, …, 1.9, 2, unknown}
- Door 2 Position: {0, 0.1, …, 0.9, 1, unknown}
Process Model
- Target Position: {0, 1, unknown}
Control Objectives
Door 2 Position
Door 2 Sensor
Position
Physical Door 2
Mech. Force
Door 2 Actuator
Door 2 Commands: {Open, Close, Stop, Unknown}
0
1
2
3
4
5
6
7
8
9
10

**Figure 2: An example HCS for the elevator system**

| | Output | Process Model | | | | Objective | *cLCons* | | |
|---|---|---|---|---|---|---|---|---|---|
| # | Elevator Movement | Motion | Floor | Door 1 Position | Door 2 Position | Target Floor | *hChk*()=1 | *hNChk*()=1 | Unreach. |
| 0 | Up | moving | 1 | 0 | 0 | 1 | - | **Yes** | - |
| 1 | Up | moving | 1 | 0 | (0,1] | 1 | **Yes** | - | - |
| 2 | Up | moving | 1 | (0,1] | 0 | 1 | **Yes** | - | - |
| 3 | Up | moving | 1 | (0,1] | (0,1] | 1 | **Yes** | - | - |
| 4 | Up | moving | 1 | 0 | 0 | 2 | - | **Yes** | - |
| 5 | Up | moving | 1 | 0 | (0,1] | 2 | **Yes** | - | - |
| 6 | Up | moving | 1 | (0,1] | 0 | 2 | **Yes** | - | - |
| 7 | Up | moving | 1 | (0,1] | (0,1] | 2 | **Yes** | - | - |
| 8 | Up | moving | (1,2) | 0 | 0 | 1 | - | - | **Yes** |
| 9 | Up | moving | (1,2) | 0 | (0,1] | 1 | - | - | **Yes** |
| 10 | Up | moving | (1,2) | (0,1] | 0 | 1 | - | - | **Yes** |
| 11 | Up | moving | (1,2) | (0,1] | (0,1] | 1 | - | - | **Yes** |

**Table 1: A subset of PHCAs of the Elevator Controller**

case study analyzes a potential risk scenario to this system because of possible threats from the deployed system's environment.

## 4.1 Identifying Consequences and Threat Surfaces

The undesirable scenario we consider for this case study is the event where the *elevator moves with an open door*. Thus, any defensive measures should ensure the *elevator only moves when the doors are closed*. Assuring or preventing this consequence would involve considering threats against three identified notional control actions, to include *Elevator Commands*, *Door 1 Commands*, and *Door 2 Commands* that respectively move the elevator car or either door.

We conduct a review of the system architecture and determine the notional control actions are handled by three control loops, each respectively managed by an Elevator Controller, a Door 1 Controller, and a Door 2 Controller. We generate a hierarchical control structure (HCS [10]) to show these control loops, shown in Figure 2. This HCS shows data flows for each numbered component and includes their input and output signals. For example, (1) Elevator Sensor provides an output signal representing the *Floor* value and (2) Elevator Actuator receives an input signal representing the *Elevator Commands*. The HCS also shows each controller's inputs and outputs, along with their internal process model variables and control objectives. Possible discrete values for all variables are included in the HCS.

## 4.2 Analyzing Potential Hazardous Control Actions

For the Elevator Controller and a Door Controller, we take discretized values of target variables deemed relevant to the consequence. We then created a table of Potential Hazardous Control Actions (PHCAs) and analyzed the discrete variable relationships that would be of interest to a threat or defender.

In the case of the Elevator Controller, we identified the following variables and their discrete values to create a table of PHCAs: output variable *Elevator Commands* = {Up, Down, Stop}; input variables *Motion* = {moving, stopped}, *Floor* = {1, (1,2), 2}, *Door 1 Position* = {0, (0,1]}, and *Door 2 Position* = {0, (0,1]}; and control objective *Target Floor* = {1, 2}. The possible values resulted 144 possible system states, 32 of which we labeled as *unreachable* because the elevator could not reach its control objective in the current state. As an example, some nodes were labeled "not possible to move Up to Floor 1 from another floor". Of the remaining 112 possible states, 42 of those states (37.5 % of possible states) resulted in $hChk(t) = 1$ as HCAs because the elevator was moving while the doors were identified as open. Another 17 states (15.2 % of the possible states) resulted in $hNChk(t) = 1$ because if the elevator did not move in a timely manner while the doors were closed, it is possible that a door could open simultaneously to lead to an HCA. Table 1 shows 12 example HCAs from our analysis to provide some clarity as to how a small subset of these target variables were analyzed.

For each Door Controller, we identified output variable *Door Commands* = {Open, Close, Stop}; input variables *Motion* = {moving, stopped}, *Floor* = {1, (1,2), 2}, and *Door Position* = {0, (0,1]}; and control object *Target Position* = {0, 1}, which resulted in 72 system states. Of these states, 24 were unreachable because of reasons such as "Open command issued while Target Position=0." Of the remaining 48 possible states, there were 15 states (31.3 % of possible states) HCAs resulting in $hChk(t) = 1$ as the doors were opened during elevator movement, and another 15 (again, 31.3 %) resulted in $hNChk(t) = 1$ and may become an HCA if the elevator moved while in that state.

If three constraints are imposed in the control logic we can eliminate all 42 $hChk(t) = 1$ states in the elevator controller and 3 out of 15 $hChk(t) = 1$ states for the door controller. These constraints are: (1) a reduction in latency should assure that the elevator's *Motion ≠ moving* when *Elevator Command = Stop*; (2) the Elevator Controller should only allow movement if both doors are closed; and (3) doors should not open when the elevator is moving. The remaining 12 Door Controller $hChk(t) = 1$ states could occur if the doors were already opened when the elevator starts moving, but ideally constraint (2) would also prevent these states. The *Floor* and control objective (*Target Floor* or *Target Position*) variables for each controller did not appear to effect event outcomes being in $hChk(t) = 1$, so we eliminate them as target attributes for this example.

## 4.3 CPS Risk Overlay Construction

With the consequence, control threat surfaces, and target variable attributes defined, we applied Algorithm 1 to generate the top of the CPS ADTree. The results of applying this algorithm are represented by the shaded nodes within Figure 3. The three control loops that were previously identified are disjointed children to the consequence node, representing three distinct actions that could lead to the consequence *elevator moves when a door is open*: (1) Elevator Controller - the elevator itself moving when a door is already open; (2) Door 1 Controller - the first floor door opens while the elevator is moving; or (3) Door 2 Controller - the second floor door opens during movement.

## 4.4 Qualitative Analysis

With the CPS risk overlay and target information, we now identify attacks, faults, and related defenses as events that may cause or prevent the scenario and then fill in the CPS-ADTree to provide a qualitative model of these relationships. We annotated each control loop node with target variable attributes to assist with event selection. We apply safety- and security-related attributes to basic actions, respectively represented in Figure 3 by a red cross or a green shield. The resulting qualitative CPS-ADTree for this example scenario is shown in Figure 3, which has 117 nodes outlining attacks, faults, and countermeasures for the scenario. The events depicted within these nodes were selected as examples to highlight the CPS risk overlay approach to identify the paths that failures must traverse and may be further refined or countered using ADTree semantics, if desired.

The CPS risk overlay does lead to many near-redundant paths that result in different physical outcomes. For example, the Door 1 Controller and Door 2 Controller have similar threats and countermeasures, as they consist of the same components in almost the same environment. In Figure 3, we abstracted away the Door 1 Controller subtree for space, simply noting it is similar to the Door 2 controller. However, if a threat impacts the components of the first control loop instead of the second, then the effect could result in a different door being opened.

As another security-related example, an attacker could target the Elevator Controller by gaining LAN access, followed by either gaining secure shell (SSH) access and then injecting commands or by injecting *Door Position* data into Modbus traffic. Conversely, to affect a Door Controller, the same attacker with the same access could target a Door Controller by injecting Elevator Controller *Motion* data into its Modbus traffic. These subtleties in attack decisions result in different physical outcomes but the same consequence.

For safety- and reliability-related issues, faults that occur within a component could lead to similar outcomes. For example, the Elevator Actuator could experience a brake failure that is countered by annual inspection. A faulty brake may lead to latency in the time between a *Stop* command action being issued and the elevator stopping and could result in the consequence. Also, the Door Controllers have flash memory components that are known to wear over time due to excessive writes, which could corrupt stored variables.

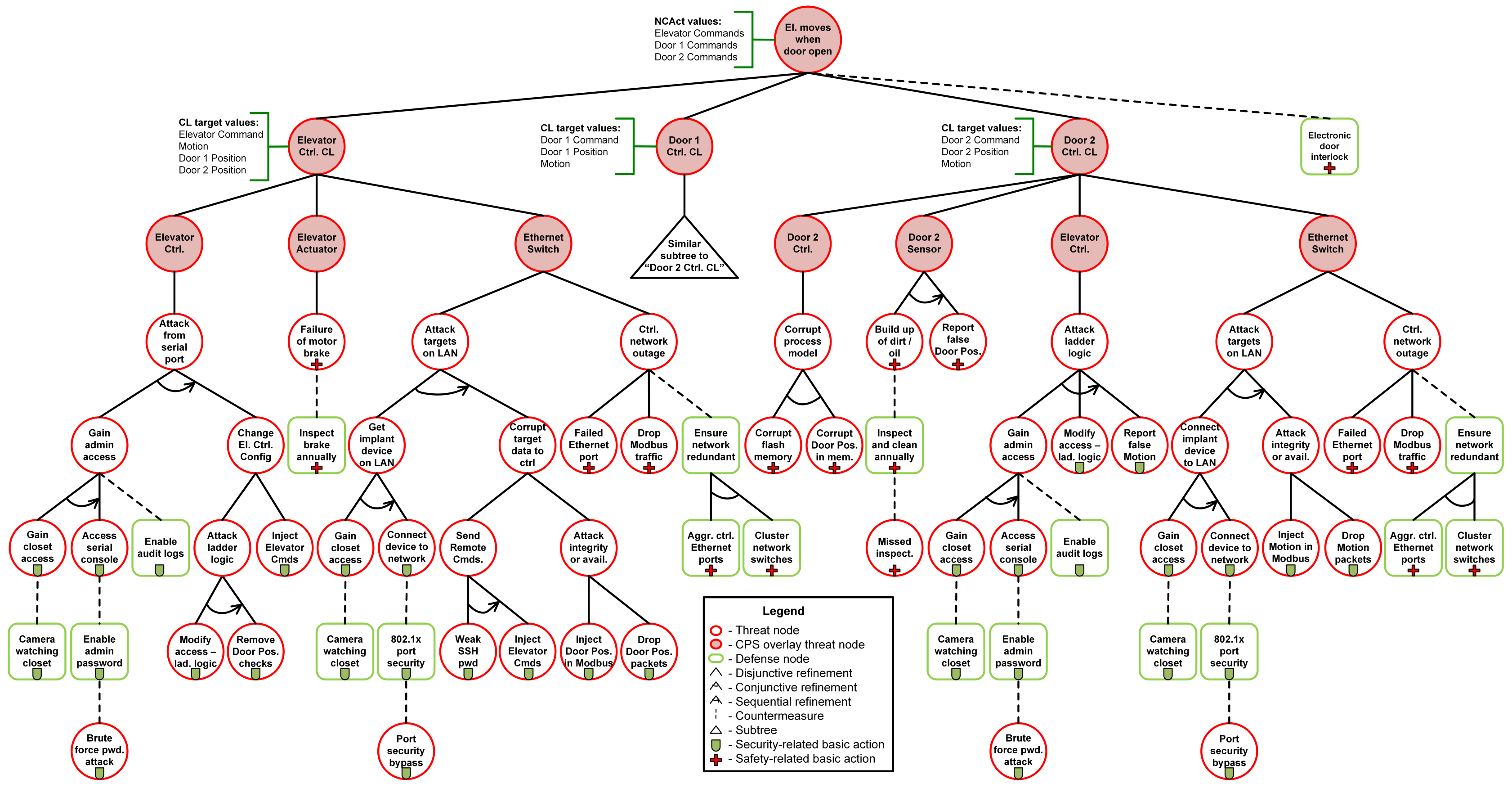


**Figure 3: CPS-ADTree for the elevator system.**

Finally, the top of the tree under the consequence, we have a countermeasure named "Electronic door interlock". Interlocks have been used as a safety mechanism in elevators for decades to ensure that doors are locked shut before the elevator car moves, so this node is a final defense mechanism against the other attack and fault events. This countermeasure also provides an example that shows how the CPS risk overlay may be refined or countered at any level. It may also be countered during the system design process if there are known attacks or faults that could impact its operation as the electric door interlock component selection and integration are further scrutinized.

## 4.5 Quantitative Analysis Discussion

The primary intent of this work was to provide a systems theoretic approach towards finding possible threat surfaces associated with a consequence and then constructing a CPS risk overlay to assist with event identification. Our approach did not change the inherited ADTree methods to quantify risk. If we were to proceed with quantitative analysis for this case study, the resulting CPS-ADTree could be assessed by assigning attributes such as costs and conducting a bottom-up analysis, as described for ADTrees in [1]. We also described how ADTrees could be reduced to attack, fault, or attack-fault trees, ensuring that the existing extensive research towards quantitative risk evaluation methods with these models is not changed by our approach.

In any case, the focus of tree-based quantitative analysis would primarily be targeting control variables in the case where $hChk(t) = 1$, resulting in some known bad system state within one of the control loops. Conversely, quantifying the adverse effects of states in $hNChk(t) = 1$ should require additional use of a system model that accounts for the time that system states occur. Such a system model is necessary because $hNChk(t) = 1$ could result in the scenario if an otherwise safe state in a control loop were to occur at the wrong time. We leave this topic for future work.

## 5 CONCLUSION

In this work we provided a systems theoretic framework for synthesizing a risk scenario to localized control decisions by converting the threat surface to a CPS risk overlay. Our method addresses a weakness of not having a system state-based description of tree-based methods, which we overcome through the use of Potential Hazardous Control Actions and control objectives. This approach allowed us to focus on analyzing relevant events for only the components that were objectively deemed within scope. We used a case study to demonstrate how different decisions targeting similar components could result in different physical results but the exact same consequence.

Our approach adheres to our own practical attack research detailed in [5] that demonstrated experimental attacks against electric motor actuation from various locations within a control loop. Here, we provide a method to augment tree-based models with those findings using systems theory to generate a CPS risk overlay using a particular case study.

## ACKNOWLEDGMENTS

This work was partially supported by the Commonwealth Cyber Initiative (CCI), which invests in cyber research and development innovation, and workforce development (cyberinitiative.org), the Department of Energy's Cyber Security for Manufacturing Automation Grant, the Cybersecurity Manufacturing Innovation Institute (CyManII), and the National Institute of Standards and Technology (NIST).

## DISCLAIMER

Commercial products are identified in order to adequately specify certain procedures. In no such case does identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

## REFERENCES

[1] Alessandra Bagnato, Barbara Kordy, Per Håkon Meland, and Patrick Schweitzer. 2012. Attribute Decoration of Attack–Defense Trees. *International journal of secure software engineering* 3, 2 (2012), 1–35.
[2] Andrew A Bochman and Sarah Freeman. 2021. *Countering Cyber Sabotage: Introducing Consequence-Driven, Cyber-Informed Engineering (CCE).* Taylor & Francis Group, Milton.
[3] Angèle Bossuat and Barbara Kordy. 2018. Evil Twins: Handling Repetitions in Attack–Defense Trees: A Survival Guide. In *Graphical Models for Security (Lecture Notes in Computer Science).* Springer International Publishing, Cham, 17–37.
[4] Jeremy Bryans, Hoang Nga Nguyen, and Siraj Ahmed Shaikh. 2019. Attack Defense Trees with Sequential Conjunction. In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE).* IEEE, Hangzhou, China, 247–252. https://doi.org/10.1109/HASE.2019.00045
[5] Matthew Jablonski and Duminda Wijesekera (Eds.). 2019. *Attacking electric motors for fun and profit.* BlackHat USA, Las Vegas.
[6] Barbara Kordy, Sjouke Mauw, Matthijs Melissen, and Patrick Schweitzer. 2010. Attack–Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. In *Decision and Game Theory for Security*, Tansu Alpcan, Levente Buttyán, and John S. Baras (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 245–256.
[7] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2012. Attack-defense trees. *Journal of logic and computation* 24, 1 (2012), 55–87.
[8] Rajesh Kumar and Mariëlle Ida Antoinette Stoelinga. 2017. Quantitative security and safety analysis with attack-fault trees. In *Proceedings - IEEE International Symposium on High-Assurance Systems Engineering.* IEEE, Singapore, 25–32.
[9] Nancy G Leveson. 2012. *Engineering a Safer World : Systems Thinking Applied to Safety.* The MIT Press, Cambridge.
[10] Nancy G. Leveson and John Thomas. 2018. *STPA Handbook.* Technical Report. Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge.
[11] Sjouke Mauw and Martijn Oostdijk. 2006. Foundations of attack trees. In *Lecture notes in computer science.* Springer, Berlin, 186–198.
[12] Bruce Schneier. 1999. Attack Trees. *Dr. Dobb's Journal* 24, 12 (December 1999), 21–29.
[13] Blake E. Strom, Joseph A. Battaglia, Michael S. Kemmerer, William Kupersanin, Douglas P. Miller, Craig Wampler, Sean M. Whitley, and Ross D. Wolf. 2017. *Finding Cyber Threats with ATT&CK™-Based Analytics.* Technical Report MTR170202. The MITRE Corporation, Annapolis Junction, Maryland.
[14] William Y Svrcek, Donald P Mahoney, and Brent R Young. 2014. *A real time approach to process control* (3rd ed ed.). Wiley, Somerset.
[15] John Thomas. 2012. *Extending and automating a Systems-Theoretic hazard analysis for requirements generation and analysis.* Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA (United States), United States.
[16] William E. Vesely, F. Goldberg, N. Roberts, and D. Haasl. 1981. *Fault Tree Handbook.* Technical Report Document NUREG-0492. U.S. Nuclear Regulatory Commission, Washington, DC.
[17] William Young and Nancy Leveson. 2013. Systems thinking for safety and security. In *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC '13).* ACM, New Orleans, 1–8.