

# Data Driven Vulnerability Exploration for Design Phase System Analysis

Georgios Bakirtzis, Brandon J. Simon, Aidan G. Collins, Cody H. Fleming, and Carl R. Elks

**Abstract**—Applying security as a lifecycle practice is becoming increasingly important to combat targeted attacks in safety-critical systems. Among others there are two significant challenges in this area: (1) the need for models that can characterize a realistic system in the absence of an implementation and (2) an automated way to associate attack vector information; that is, historical data, to such system models. We propose the cybersecurity body of knowledge (CYBOK), which takes in sufficiently characteristic models of systems and acts as a search engine for potential attack vectors. CYBOK is fundamentally an algorithmic approach to vulnerability exploration, which is a significant extension to the body of knowledge it builds upon. By using CYBOK, security analysts and system designers can work together to assess the overall security posture of systems early in their lifecycle, during major design decisions and before final product designs. Consequently, assisting in applying security earlier and throughout the systems lifecycle.

**Index Terms**—Cyber-physical systems, security, safety, model-based engineering.

## I. INTRODUCTION

It has been estimated that 70% of security flaws are introduced prior to coding, most of which are due to the traditional practice of application developers sharing and reusing third party, legacy software—that is assumed to be reasonably secure and trustworthy. These flaws usually end up in the application software and not, as might be expected, in network-based software [1, 2, 3]. Most security flaws are introduced as early design or development decisions.

Both the academic and practicing cybersecurity community agree that security engineering and analysis as a full lifecycle practice, especially early in the design process allows better awareness and leverage at managing the challenges surrounding the unintentional introduction of security flaws into complex systems. This is especially important in the domain of cyber-physical systems (CPS), where the exploitation of software flaws and hardware weaknesses—introduced by either importing software of unknown pedigree, incomplete security specifications, or general unawareness of security characteristics of given software, firmware, and/or hardware—can lead to unforeseen physical behaviors that have consequences in terms of safety, loss of vital service, and other societal impacts.

As modern CPS evolve into tightly integrated, extensible, and networked entities, we significantly increase the attack surface of these systems. CPS now routinely employ a wide variety of networks, for example, cloud, mobile services, industrial, internet of things to realize a range of applications

from real time data analytics to autonomous vehicles control. The use of extensible operating systems and software to update code through loadable device drivers enhances productivity, but it exposes the system to considerable risks from attack injections.

With these insights and observations, we posit that secure system design and deployment requires (1) planning for cybersecurity from the outset as a strategic lifecycle activity, and (2) taking the attackers perspectives to best understand how to defend a system from threats and exposing weaknesses before they become vulnerabilities. To achieve this goal methods and tools are needed to allow security assessment throughout the systems lifecycle and especially at the concept development phase, where decision effectiveness is highest [4, 5].

In recent years, a promising and rapidly growing approach to enhancing awareness and managing challenges of cybersecurity flaws in evolving complex CPS is model-based engineering [6]. Model-based analysis is firmly entrenched in safety, dependability, and reliability engineering world as evidenced by such standards as IEC 61508 and ISO 26262, however model-based engineering is a late comer to security [7].

Models are generally treated as *living documents* maintained to reflect design choices and system revisions. These models can be a valuable resource for the security specialist by providing what IT professionals consider the “what’s”; that is, the rationale behind design choices and not simply the resulting architecture of a system [8].

An additional benefit of model-based security is it tends to look at security from a strategic point of view, which means it attempts to secure a system based on its expected service. Rather than beginning with tactical questions of how to protect a system against attacks, a strategic approach begins with questions about what essential services and functions must be secured against disruptions and what represents unacceptable losses. This is critical for CPS where losses or disruptions to service can have dire societal or safety impacts [9, 10].

However, one of the major impediments to effectively transition security assessment into the model-based engineering realm has been associating system models to applicable attack vectors. These models reside in a higher-level of abstraction than what is typically present in cybersecurity analysis. Our aim is to use the model to drive the attack vector analysis in the design phase. There are two things necessary to achieve this congruence: (1) understand the data available to security researchers and decide on which of those can inform early on and (2) capture lower-level information in the model, such that it can be used to associate the available data with the model. Such an approach bridges the gap between existing curated attack vector information and models of systems. Indeed,

G. Bakirtzis and C.H. Fleming are with the University of Virginia, Charlottesville, VA USA. E-mail: {bakirtzis,fleming}@virginia.edu

B.J. Simon, A.G. Collins, and C.R. Elks are with Virginia Commonwealth University, Richmond, VA USA. Email: {simonbj,collinsag,crelks}@vcu.edu

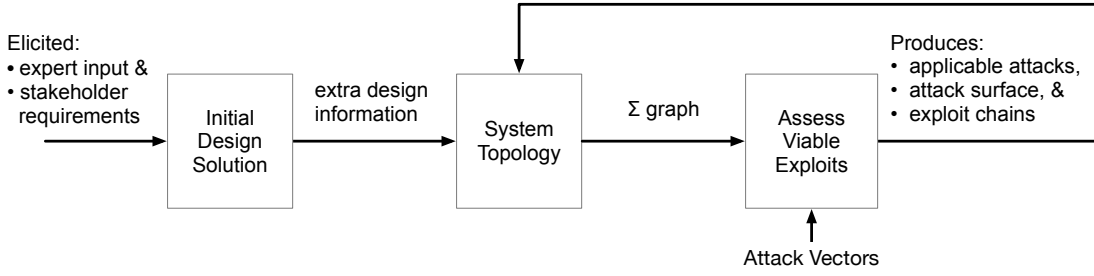


Fig. 1. CYBOK depends upon stakeholder requirements and expert input to construct the initial design solution. Then, extra design information is added by a system designer. CYBOK takes as input the graph representation of that design and descriptions of attack vectors to produce all applicable attacks throughout the topology of the system, the attack surface of the system, and the exploit chains traversing the system topology.

this paper presents one answer to examining cybersecurity concerns in the model-based engineering setting.

Towards this goal we have previously [11] presented a CPS model that includes a schema with extra design information to manually associate attack vector data describing attack patterns, weaknesses, and vulnerabilities. The previous work proposed just a model, not an algorithmic solution to the vulnerability exploration problem. To explore the large amount of data compiled by security professionals, it is helpful to associate attack vectors algorithmically. This is precisely the topic of this paper.

**Contributions.** The contributions of this work are:

- an algorithmic implementation, called cybersecurity body of knowledge (CYBOK), that accepts as input such models and produces:
  - a component-wise attack vector analysis using attack vector data,
  - a notion of attack surface, which only depends on the model, and
  - all exploit chains applicable to subsystems of the system model and
- a demonstration of the method on an unmanned aerial system (UAS).

## II. MODEL-BASED SECURITY ANALYSIS

Model-based security analysis is a relatively new field that attempts—as the name implies—to understand system threats through the use of models. In this paradigm, models are used either as an augmentation to other security strategies during deployment or as evidence to support design decisions early in the systems lifecycle. However, most current models are probabilistic in nature and, therefore, require a ground truth. These models also heavily depend on the modelers expertise and experience. Any such model captures exactly that expertise such that it is communicated to other stakeholders. To our knowledge, models used at the design phase have not achieved fidelity with security data collected and otherwise used in already realized systems, which would consist of an important addition to defending against increasingly sophisticated threats.

But why is that? To understand the difficulty of finding vulnerabilities in system models—instead of a deployed product—it is important to first define the difference between bugs, vulnerabilities, and exploits. Successful exploits take

advantage of flaws (either serious design flaws or unexpected system behavior that is implementation specific). These flaws in the system are called bugs. However, not all bugs are vulnerabilities. Only a subset of bugs that can lead to exploitation are vulnerabilities. This notion leads to the first problem that is addressed in this paper.

**Problem 1.** Vulnerabilities are explicitly found at the level of code or hardware. However, to address system security early in the design cycle, there need to be methods that can identify potential vulnerabilities before code development.

To bridge the gap between models and realized solutions requires constructing an initial design of the system. This design needs to include both the *what's*, the components of the system, for example GPS, and the *how's*; that is a particular hardware, firmware, and software solution that implements some desired function. Additionally, any such model needs to include the interaction between components as is defined by their communication and data transfer.

One way to fulfill those requirements is to model CPS as a graph of assets but with added information in the form of descriptive keywords. This is a reasonable and appropriate model as it pertains to security analysis. Attackers typically think in terms of graphs, through a series of increasing violations based on concepts of connectivity, reachability, and dependence, not in lists of assets as—most commonly—defenders do [12].

In addition, this model must include extra information in the form of keywords that augment the model, resulting in a system model that captures the choices a designer is considering about hardware, firmware, and software. These augmentations can be done without overly specific details about its final implementation. This is a key feature that reflects how designs evolve in the construction of a system, where choices of specific hardware and software are done early in the development cycle. Furthermore, the ease of changing those keywords to describe a functionally equivalent system allows for modeling flexibility that is not available after code has been written and designs are finalized.

Formally, an architectural model of a CPS can be captured in a graph,  $\Sigma \triangleq (\mathcal{V}, \mathcal{E}, \mathcal{D})$ , where,

$$\mathcal{V} \triangleq \{v \mid v = \text{a system asset}\},$$

$$\mathcal{E} \triangleq \{e \mid e = (v_i, v_j); v_i, v_j \in \mathcal{V} \text{ dependent assets}\}, \text{ and}$$

$$D \triangleq \{d \mid d = (w_1, w_2, \dots, w_n) \text{ descriptive keywords}\}.$$

In order to extract and use the descriptive information; that is, the extra keywords, from a given vertex or edge, we define the descriptor function,  $desc : \mathcal{V} \cup \mathcal{E} \rightarrow D$ .

The above definitions lead to the first practical challenge, which is to determine if a model is sufficient for security assessment (Fig. 2). It is important at these early design stages for a system model to sufficiently describe functionally complete system—by adding hardware and software information that, if put together, implements the desired functional behaviors expected from the system.

Any model used for security analysis contains two main challenges. The first challenge, is the amount of data associated with the model. When using a realized system it is possible to mine all possible configuration settings including software and firmware versions. In the absence of the implementation such information can still be reflected in a modeling setting but it requires significant modeling cost in terms of time and expertise. It can also be less informative at the design phase than a more general model because a slight change in versioning will hide a class of weaknesses and vulnerabilities. The second challenge, is the level of abstraction the model resides in. No model is a direct reflection of a realized system but any model needs to be specific enough to be informative. This is a difficult task and largely depends on the given abstraction set overall by the modeling process as well as the expertise of the modeller.

These are precisely the challenges that the added keywords address. By changing the specificity and amount of keywords, we change the overall fidelity of the system model,  $\Sigma$ . It is through that *extra* design information that our solution, CYBOK, is able to take the graph of a system model and map applicable attacks from security databases (Fig 1). While there may be a number of different criteria for selection, in previous work we have found that the extra design information can be categorized through the following practical schema: operating system, device name, communication, hardware, firmware, software, and entry points [11]. Each of the categories is expected to contain a string of keywords,  $d \in D$  that collectively describe a given system solution. A given category can also duplicate the descriptive keywords present in another category or simply contain the null set,  $\emptyset$ .

The fidelity of the model is still based on the choices of the modeller. On the one side of the *model sufficiency spectrum* there are designs that are too general and do not contain information about the system that would aid in determining security posture. On the other side of the spectrum there are designs that are too specific, to the extent that the effort to create and potentially modify is equivalent to constructing an actual implementation of the system and its functionality. Such systems are complete but impractical. There is a spot in the middle of the spectrum, where the information contained in the model can provide a reasonable idea about the system's threat space without being so detailed it is inflexible and costly to construct and maintain throughout its lifecycle.

A perhaps less obvious but equally important challenge refers to the information necessary to associate the model,  $\Sigma$ ,

		Number of Attributes	
		Low	High
Specificity of Attributes	Low	Both unrealistic and insufficient in detail to provide attack vector results	Insufficient detail to provide attack vector results
	High	Unrealistic—not representative of the actual system—model	Increases modeling effort and complexity (ideal but possibly prohibitive)

Fig. 2. The fidelity of the attributes describing a CPS has to associate to the attack vector information (reproduced from Bakirtzis et al. [11]).

to potential attack vectors.

**Problem 2.** How can we associate vulnerability, weakness, and attack pattern information that is intended to be used by security analysts to a model?

This problem is difficult because of the way security experts record vulnerabilities. While several attempts have been made to standardize the form of an attack vector entry, the current situation is such that the different databases are based on a different schema, because they rely on the deduction and inference capabilities of a human. This means there is no straightforward approach to feeding that data into a machine to automatically find those mapping.

Our solution is based on the set of descriptors,  $D$ , present in the model and using standard practices from natural language processing to deconstruct and associate the contents of an attack vector entry—in the form of text—to the models keywords,  $d \in D$ . This requires a separate set of attack vector entries,  $\mathcal{AV} \triangleq \{av \mid av = \text{a set of stemmed words}\}$ . Therefore, the fundamental problem that CYBOK attempts to solve is then formalized as the function,  $associate : desc \rightarrow \mathcal{AV}$ .

By doing so, the problem is reduced to associating keywords describing the system (which exist within the model) to stemmed words describing attack vector information (which are constructed using the contents of the database entries and natural language processing).

Applying security consideration to model-based engineering is difficult for several reasons. Three of the most important difficulties are: the curation of information (both from the model and the attack vector databases), the intuition surrounding the fidelity of the system model, and the development of an algorithmic approach that allows for filtering through a large number of attack vectors produced at the design phase.

Finding attack vectors for individual vertices or edges

overcomes these challenges. However, it can be daunting to see the big picture when confronted with such a larger amount of data. Security professionals frequently use complimentary metrics to understand the overall security posture of a given system.

Two such useful metrics for security analysis are:

- 1) The *attack surface* captures all the entry points into a given system [13].
- 2) The potential for further spread; that is, further violations, after an element of the attack surface has been compromised, known as an *exploit chain*.

**Proposed Solution.** In general, CYBOK is an algorithmic solution that takes as input a sufficient system model (of  $\Sigma$  form) to associate to the body of knowledge of attack vectors (of  $\mathcal{AV}$  form). By knowing the associated attack vectors it then produces security metrics only based on the model; that is, the attack surface of the system model and the exploit chains for a particular element of the model.

**Intrinsic Limitations.** Model-based security analysis is grounded on early design information. This early design information is usually incomplete and abstracted with respect to the final design solution. This leads to result spaces; that is, associated attack vectors, that are significantly larger than when analyzing a realized system. Navigating through the results can be challenging for systems engineers that are not familiar with security practices. Our aim is to provide a framework in which security analysts and systems engineers work synergistically to understand both necessary design decisions (that might affect potential security mitigations) and security considerations (that might affect the design of the system).

An additional limitation is the fidelity of the model. The model can only be as good as the person who is modeling the system. Therefore, a poorly constructed model might mislead instead of providing insight into the security posture of the system.

### III. SYSTEM MODEL

To address the challenges in the previous section and construct a sufficient model with respect to vulnerability analysis, first we must elicit information from the stakeholders. The stakeholders of the system include the owners of the eventual system, the system designers, the safety engineers, and the security analysts. While it is outside the scope of this paper to address the systematic process in which such information is elicited (see Carter et al. [14] for further details on the topic), it is important to place CYBOK within its larger framework. Without this framework it would not be possible to have complete or correct information to apply vulnerability exploration this early in the system's lifecycle. Based on this elicitation, an initial design solution is modeled in the systems modeling language (SysML).

SysML uses visual representations to capture the system design process through objects. The main benefit of SysML is that it presents the same information in different views, which allows the same system to be modeled based on its

requirements (through the requirements diagram), through its behavior (through, for example, the activity and/or state machine diagrams), and/or through its architecture (through block definition diagram (BDD) and/or internal block diagram (IBD)).

To model CPS architectures in SysML the system structure is captured as a set of BDD and IBD. The BDD view of the system shows the composition of the system. The IBD view refines those compositions to interconnections within the system and how those interconnections compose the system behavior.

However, each element of the system model is described by a standardized schema as presented by Bakirtzis et al. [11]. It is, therefore, not necessary to capture this model in SysML. This model is flexible to design changes and has supported vulnerability analysis in a manual setting. In this work we use the modeling methodology to support automated vulnerability analysis.

Security specialists usually construct the following information implicitly through expertise. To automate this task this implicit information needs to be captured explicitly in the model. This information will also assist in constructing a living document describing the *what's* of those choices. To recap, the schema is composed by the following categories that describe each system element:

- operating system,
- device name,
- communication,
- hardware,
- firmware,
- software, and
- entry points.

It is through that *extra* design information—gathered by eliciting stakeholder information and inspecting design documentation—that CYBOK is able to take the graph of a system model and map potential attacks from databases (Section IV). This is done by using the key terms presented in the schema for each element and checking if they are present in the documents composing the databases.

Reasoning in terms of the diagrams has several benefits during the design process that hold for security analysis in general (see Oates et al. [15]), which is the benefit of starting with a SysML model instead of its graph representation. This is less true for matching attack vectors to the model. The exporting of models in a standardized format is, therefore, beneficial. The translation of IBD diagrams into a graph is encoded into GraphML—a simple XML format that is widely used to import and export graph structures [16].

The two models—i.e., the visual representation in SysML and the graph structure—must be isomorphic. This means that the transformation between the SysML model and the GraphML representation must not change the model of the system. To achieve an isomorphic transformation we apply a model transformation on the IBD model. This transformation produces a sufficient graph model for security analysis (Fig. 5 in Section VI-B).

**Property 1 (Model Transformation).** An INTERNAL BLOCK DIAGRAM is the graph  $I \triangleq (\mathcal{V}, \mathcal{P}, \mathcal{D})$ , where  $\mathcal{V}$  is the set of vertices of  $I$  and  $\mathcal{P}$  is the set of ports of  $I$ . Further,  $\mathcal{V}$  represents the assets of a cyber-physical system,  $\mathcal{P}$  the dependence between assets, and  $\mathcal{D}$  the descriptors of each asset. Therefore, the graph  $I$  is isomorphic to our system graph  $\Sigma$ ; that is to say  $I \cong \Sigma$ .

This system model represented as a graph allows us to think similarly to attackers and to construct connections that would not be obvious if treated as simple decoupled components. The graph of the system model assists with not only finding vulnerable subsystems individually but also with finding the attack surface of the system and composing exploit chains. Exploit chains are a subset of attacks that could traverse though the system model graph and elevate the impact to deteriorate system behavior. All exploit chains start at elements of the attack surface. The attack surface,  $\mathcal{AS}$  is composed by all vertices that an attack from the databases is found to be potentially applicable at the entry point description.

In particular the graph model allows us to define the attack surface and exploit chains over the system model  $\Sigma$  in a straightforward manner.

**Definition 1 (Attack Surface).** We define the attack surface of a system model,  $\Sigma$  as  $\mathcal{AS} \subseteq \mathcal{V}$ , which is composed by all vertices that can be entry points into the system and allow the attacker to cause further spread within the system structure.

**Definition 2 (Exploit Chain).** To construct the exploit chain we define a function,  $paths : \mathcal{AS} \times t \times \Sigma \rightarrow \mathcal{P}$ , where  $\mathcal{AS}$  the sources of all paths and  $t$  a used specified target and  $\mathcal{P}$ , the set of all simple paths from the source to the target over  $\Sigma$ . Then to construct a single exploit chain,  $ec \in \mathcal{EC}$ , the set  $\mathcal{P}$  is filtered by checking if every vertex and edge within each individual path associates to some attack vector from  $\mathcal{AV}$ .

#### IV. ATTACK VECTOR DATASET

CYBOK is composed by several databases to address two main challenges. The first challenge is finding applicable attack vectors based on a system model. The second challenge is to present a reasonable amount of data to the security analyst, such that they can erect barriers or add resilience solutions to strengthen the design of CPS using an evidence-based approach.

To address these challenges, CYBOK incorporates three collections curated by the MITRE corporation: CVE [17, 18], CWE [19], and CAPEC [20]. CVE is the lowest level of attack vector expression, defining tested and recorded vulnerabilities on specific systems. CWE presents a hierarchy of known system weaknesses at different levels of abstraction, from which exploits can be derived. Finally, CAPEC provides a high level view of attacks against systems at varying levels of abstraction, in the form of a hierarchy organized by the goal or mechanism of each attack. These three collections also include relationships to one another (Fig. 3). Formally CAPEC, CWE, and CVE construct the set of attack patterns, weaknesses, and vulnerabilities  $A \times W \times V \cong \mathcal{AV}$ .

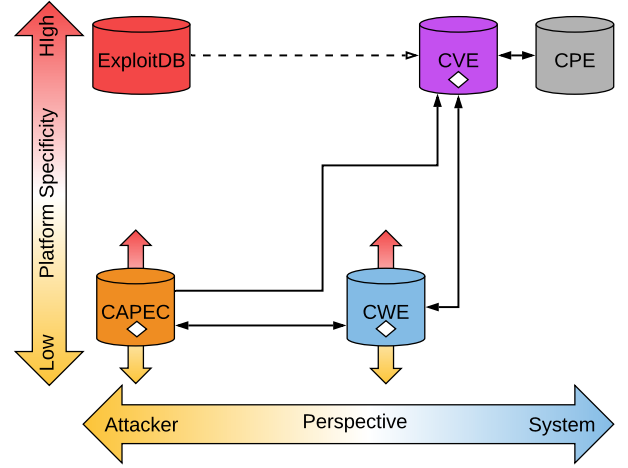


Fig. 3. The collection of the most common open attack vector datasets and their interconnections in the sense of topical relationships with regard to attacker and defender perspectives and level of specificity they contain about platform information. Edges denote which datasets have explicit relationships with one another. CYBOK only uses a subset of these databases marked by a  $\diamond$ .

Other known datasets include the MITRE Common Platform Enumeration (CPE) [21] and the Exploit Database (exploit-db) [22]. The former includes information that is platform specific, including particular versions of software that a particular exploit is associated with. The latter provides samples of exploits for given CVE entries. The reasons for not including these two datasets is because CPE requires knowing the specific versions of software that are going to be on the system—which are not known at design phase—and exploit-db requires a realized system to test exploits against.

Using CVE entries has the benefit of finding additional attack vectors because the specificity of their descriptions may more closely associate to the descriptions in the model than those of CAPEC and CWE. When CVEs are matched to the system model, CYBOK uses that information to abstract upwards towards the weaknesses and the attack patterns. This is especially useful in the case where multiple CVE entries are associated with a subsystem that has the same associated weakness or attack pattern.

In general the CWE and CAPEC abstractions are more useful to designers over CVE entries because CVEs are too specific to be useful at the design phase. For example, being aware of a vulnerability in a specific version of software is less illuminating than knowing that a specific class of software bugs might consist of a vulnerability in the implementation of the system—and therefore can construct more concrete requirements or define specific mitigations. On the other hand, a number of applicable attack vectors from the model are going to reside in CVE. At the same time, CVE contains a significantly larger number of entries than both CAPEC and CWE ( $\sim 100,000$  vs.  $\sim 1500$ ), meaning the addition of CVE entries will explode the number of results for a given system  $\Sigma$ . Therefore, all three are needed: CVE entries to be more thorough and complete in analyzing the system model and CAPEC, CWE to abstract to useful information to system designers.

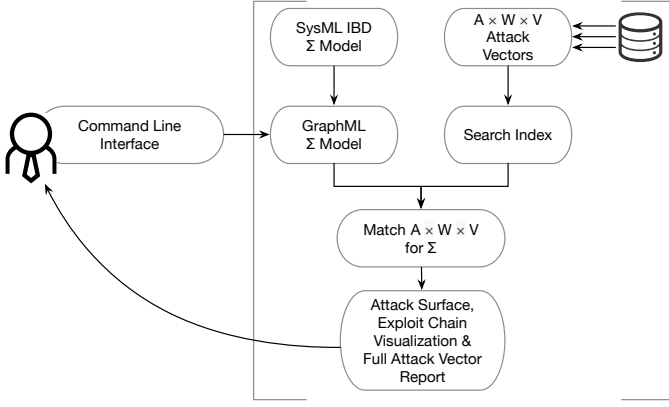


Fig. 4. The architecture of CYBOK is modular, meaning that each of the functions can be replaced without needing to interfere with other functions.

While the three selected collections are complete in relation to the sufficient model, CVE is certainly not exhaustive. There are certainly instances where companies or government agencies curate more exhaustive databases from their non-disclosed findings. In those cases, the design of CYBOK can be extended to use the information contained in these private databases.

## V. ARCHITECTURE

The overall architecture of CYBOK is designed to be modular and robust with respect to potential architectural changes (Fig. 4). For example, the specific implementation of searching can be changed without needing to change the rest of the core tool functionality.

### A. Data Extraction, Preprocessing, & Indexing

The first step to associating models to attack vector databases is to extract and preprocess the information contained in several individual databases. An automated mechanism for downloading the latest set of data is built into CYBOK because of the CVE, CWE, and CAPEC update cycle.

Specifically, CAPEC is refactored and potentially updated every six months to a year, CWE is extended every one month to three months, and CVE adds new entries daily. By doing automatic updates, the analyst is sure to have the latest set of data that might inform about new system violations.

All database files are encoded in a standard xml file format. The steps that follow after updating the data files include preprocessing and constructing the search index to be used to associate system models to attack vectors.

The preprocessing step extracts the name of the database, the identification number, name of attack vector, associated attack patterns (if any), associated weaknesses (if any), associated vulnerabilities (if any), and the contents; that is the description, for each entry. These consist of the full search index schema; that is, all the information necessary to construct  $A \times W \times V$ .

After preprocessing CYBOK keeps a persistent record of all attack vectors,  $\mathcal{AV}$ , by constructing a search index through the schema defined above. This allows for efficient information

retrieval of all attack vector information and avoids having to rebuild  $\mathcal{AV}$  for each new search query.

### B. System Models as Graphs

Graph structures provide an important view in a computing system, and can extend the notion of violation to more than just a singular view of components. Indeed, the violation of a single component by an attacker might not be detrimental to the systems expected service. However, this component might be connected to other critical infrastructure. Therefore, this singular component could be a point of lateral pivot for an attacker. This in turn can cause significant malfunction during operation with catastrophic consequences. This is how attackers operate and, therefore, reasoning in graphs of assets provides an attacker's view to defenders. For this reason, CYBOK views the system topology; that is, the design artifact, as a graph.

### C. Finding Applicable Attack Vectors from a System Model

---

#### Algorithm 1 Finding attack vectors

---

```

1: function ASSOCIATE( $\Sigma$ ,  $\mathcal{AV}$ )
2:    $\mathcal{R} \leftarrow []$ 
3:   for all  $desc(v) \in \mathcal{V}(\Sigma) \wedge desc(e) \in \mathcal{E}(\Sigma)$  do
4:     for all  $d \in \mathcal{D}$  do
5:       for all  $av \in \mathcal{AV}$  do
6:         if  $d \in av$  then
7:            $\mathcal{R}.append(\{v \vee e, d, av\})$ 
8:   return  $\mathcal{R}$ 

```

---

To find applicable attack vectors, CYBOK extracts the descriptive keywords that define each vertex and edge. Then, using the descriptive keywords of the system CYBOK looks at all attack vector entries from  $\mathcal{AV}$  to associate the descriptive keywords. A list of results is returned for the full system model,  $\Sigma$ , including the vertex or edge the attack vector can exploit, the descriptive keyword,  $w_i$ , that produced the attack vector, and the attack vector itself (Algorithm 1).

The search functionality of CYBOK currently uses a compound word filter. Other candidates for applying the searching include n-grams and Shingle filter.

### D. Finding Attack Surface Elements

---

#### Algorithm 2 Finding attack surface elements

---

```

1: function ATTACKSURFACE( $\Sigma$ ,  $\mathcal{R}$ )
2:    $\mathcal{AS} \leftarrow []$ 
3:   for all  $entry\_points(desc(v)) \in \mathcal{V}(\Sigma)$  do
4:     if  $\{v, d, \_ \} \in \mathcal{R}$  then
5:        $\mathcal{AS}.append(\{v, d\})$ 
6:   return  $\mathcal{AS}$ 

```

---

CYBOK views the attack surface as any vertex that has an associated attack vector specifically at the entry point. It constructs this set by going through all vertices and checking if a descriptive keyword,  $entry\_points(w_i)$ , associates to an applicable attack vector (Algorithm 2).

## E. Finding Exploit Chains

---

### Algorithm 3 Finding exploit chains

---

```

1: function EXPLOITCHAINS( $\Sigma$ ,  $\mathcal{R}$ ,  $\mathcal{AS}$ ,  $t$ )
2:    $\mathcal{EC} \leftarrow []$ 
3:   for all  $as \in \mathcal{AS}$  do
4:     for all  $p \in paths(as, t, \Sigma)$  do
5:       for all  $v \in p \wedge e \in p$  do
6:         if  $\{v \vee e, \_ , \_ \} \in \mathcal{R}$  then
7:            $admissible\_path \leftarrow \top$ 
8:         else
9:            $admissible\_path \leftarrow \perp$ 
10:        break
11:       if  $admissible\_path == \top$  then
12:          $\mathcal{EC}.append(\{p\})$ 
13:   return  $\mathcal{EC}$ 

```

---

Exploit chains are paths from a source to a target that contain violation for every vertex or edge in that path. CYBOK finds exploit chains from all elements of the attack surface,  $\mathcal{AS}$  to a user input target,  $t$  (Algorithm 3). These paths are not necessarily the most efficient or direct paths from the elements of the attack surface to the given target. This is because it is often the case that attackers move laterally from the attack surface to a specific target without having full observability of the system. This way the analyst can be aware of all paths that are valid based on the system model and be better informed about potential mitigations.

Furthermore, not all paths from  $\mathcal{AS}$  to  $t$  are admissible. Admissible exploit chains require each vertex and each edge in that path has produced at least one result from  $\mathcal{AV}$ . Otherwise the path is not fully exploitable based on evidence and, therefore, does not consist of an exploit chain under this definition.

## F. Visualizations

To facilitate the analysis of results, CYBOK includes three main visualizations: (1) the system topology, (2) the system attack surface, and (3) the system exploit chains. This is an important feature of CYBOK because it allows both security analysts and system designers to project how exploits propagate over the system model,  $\Sigma$ . This way, they can be better informed about potential mitigation strategies. For example, changing the definition of a single element to one that has no recorded attacks might significantly increase the security posture of the overall CPS.<sup>1</sup>

Moreover, a full GUI is developed based on this methodology to implement further interactivity functions on top of CYBOK [23]. This is a natural progression of CYBOK since in-depth analysis requires the analyst to interact with the data through interactivity functions, for example, filtering, to facilitate effective exploration of the diverse types of data input and output to and by CYBOK [24].

<sup>1</sup>We assume that a component with no recorded attacks is less susceptible to exploitation over one that has a large number of recorded attacks.

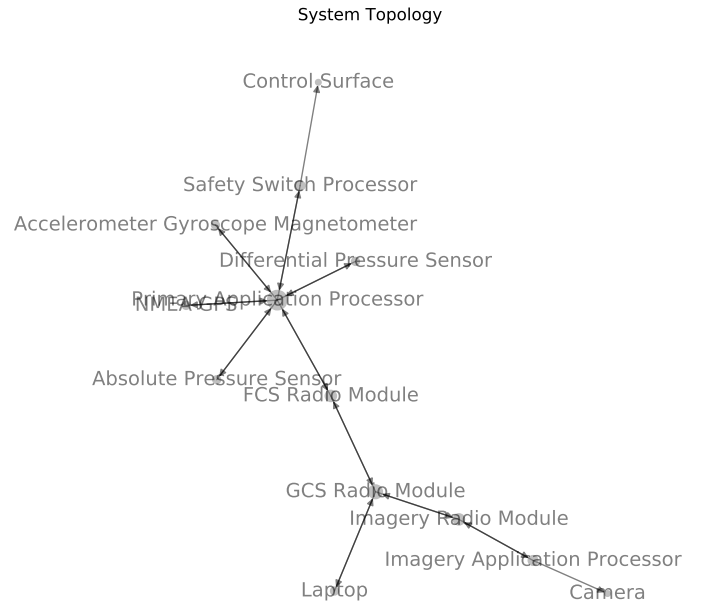


Fig. 5. The system topology,  $\Sigma$ , shows a static view of the system model.

## VI. EVALUATION

To evaluate CYBOK we will discuss in some detail the vulnerability analysis of one potential design solution for a UAS that contains a full set of descriptors. There is ongoing work in applying CYBOK to several other systems in the military and nuclear power domain [25].

### A. System Model

While modular approaches to flight control systems (FCS) have been demonstrated to provide flexible choices in hardware [26], it is not currently possible to assess the security of one design over another before building the system. By using models of systems it is possible to assess several system designs and provide evidence over the use of one hardware solution over another. In this work one such hardware solution is evaluated—through its system model (Fig. 5)—and present the evidence that stems from assessing the model’s security posture using CYBOK.

The potential design solution present in this paper uses several XBee radio modules to communicate between components, Dell Latitude E6420 ground control station (GCS) laptop, an ARM STM32F4 primary application processor, a BeagleBone Black imagery application processor, an ARM STM32F0 safety switch processor, MPU9150 accelerometer, gyroscope, and magnetometer, MS4525DO differential and absolute pressure sensors, a GoPro Hero5 camera, and an Adafruit Ultimate GPS. This information is part of the descriptive keywords captured in the vertices of the model. Further information is given for both vertices and edges to drive this analysis per the schema above (Section II).<sup>2</sup>

### B. Example Analysis

SysML is used as the modeling language and tool because it is often familiar to Systems Engineers. However, CYBOK

<sup>2</sup>The model is publicly distributed [27].

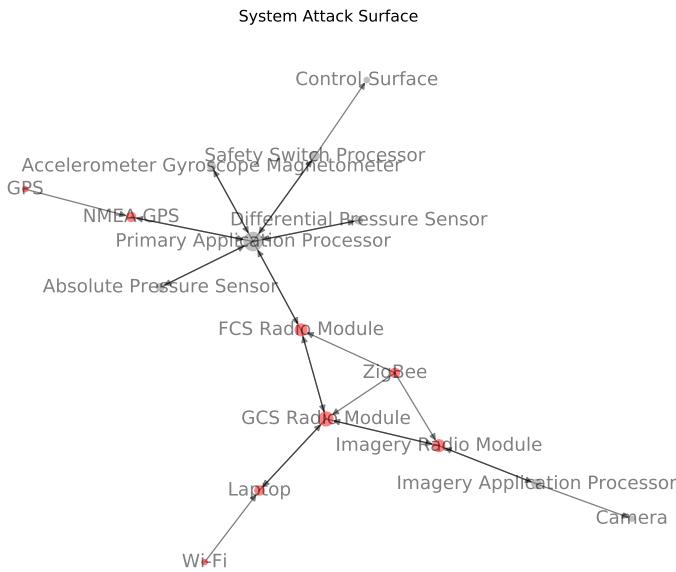


Fig. 6. The attack surface,  $\mathcal{AS}$ , extends the system topology,  $\Sigma$ , by adding in the descriptive keywords at the entry point that associate to attack vectors.

is not restricted to SysML models, it merely requires a graph representation of the system that includes extra design information (see Bakirtzis et al. [11] and `graphml_export` [28] on how this translation is achieved in practice). Inputting the UAS system topology to CYBOK first constructs the attack surface (Fig. 6). The attack surface is extended to show all the descriptive keywords at the entry points that attack vectors are found. For example, by inspection the use of the XBee module with the ZigBee protocol for all three radio modules can be problematic because an attacker can exploit the system remotely. Other such entry points have a different degree of potential exploitation. It is unlikely that the GPS will be violated but attacks for GPS exist and, therefore, are reported by CYBOK. Additionally, the analyst might be aware of hardening techniques on the Wi-Fi network used by the GCS laptop. Consequently, the analyst might decide that they consist of no threat to the systems mission.

From this initial understanding of the systems security posture (through its composition and attack surface) an analyst can further interrogate the model by finding all the potential exploit chains from the attack surface elements to the primary application processor. This is because violation of the primary application processor will cause full degradation of system functions and, therefore, full mission degradation overall. Specifically, an analyst might want to examine a potential exploit chain stemming from the XBee element of the attack surface. By providing a target,  $t$ , CYBOK finds the admissible paths and, therefore, exploit chains from the imagery radio module to the primary application processor. This path is admissible if each vertex *and* each edge within that path has produced evidence; that is, attack vectors.

Examining the results produced by CYBOK we find the following three associated entries: CAPEC-67 “String Format Overflow in syslog(),” CWE-20 “Improper Input Validation,” and CVE-2015-8732 a specific attack on the ZigBee protocol used by XBee that allows remote attackers to cause a denial

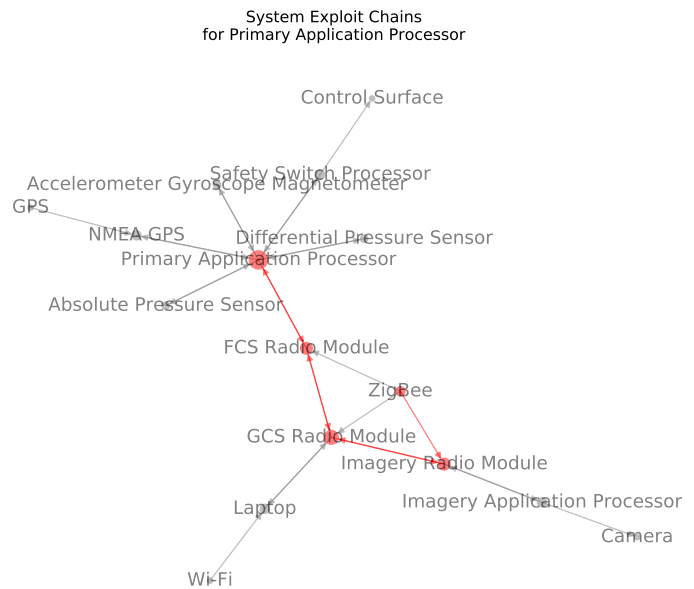


Fig. 7. Exploit chains,  $\mathcal{EC}$ , show a possible lateral paths an attacker might take over the system topology to reach a specific system element. This is but one example of such exploit chain from the attack surface,  $\mathcal{AS}$ , to some target  $t$ —in this case the primary application processor.

of service (DOS) via a crafted packet. Further, for the edges from the radio module to the primary application processor there are the following attack vectors produced by CYBOK, CVE-2013-7266, which is a specific attack that takes advantage of not ensuring length values matching the size of the data structure, CWE-20 “Improper Input Validation,” CWE-789 “Uncontrolled Memory Allocation,” CWE-770 “Allocation of Resources without Limits or Throttling,” and CAPEC-130 “Excessive Allocation.” Finally, the primary application processor uses the I2C and RS-232 protocols to communicate with the rest of the hardware (these are descriptive keywords contained in the edges of the graph), which produce the following, CAPEC-272 “Protocol Manipulation” and CAPEC-220 “Client-Server Protocol Manipulation.” All this information is used as evidence for the feasibility of one exploit chain from the attack surface to the primary application processor (Fig. 7).

By projecting the attack over the system structure it is evident when the same attacks are applicable to several parts of the system. This is important because attackers contain a specific skill set and they do not usually deviate from it if not necessary.

Additionally, CYBOK allows flexible “what-if” analysis by changing the descriptive keywords in the model. For example, by changing the radio module definition from XBee using the ZigBee protocol to some other radio module offered in the market might exclude it from the attack surface. Since a larger attack surface implies more access points and, therefore, a less secure system an analyst might decide to propose changing the design of the system.

A full analysis consists of first identifying the important elements of the system; that is, the assets that might require protections. This might be informed from the outputs produced by CYBOK or from expert input and information elicitation. Then, of filtering the large space of attack vectors that asso-



TABLE I  
A FRAGMENT OF RELEVANT RESULTS FOR THE UAS AS PRODUCED BY CYBOK.

Model Element	Attack Vector	Description
Radio Modules	CVE-2015-6244 CWE-20 CAPEC-67	Relies on length fields in packet data, allows attacks from crafted packets Improper input validation String format overflow in syslog()
NMEA GPS	CAPEC-627 CAPEC-628	Counterfeit GPS signals Carry-Off GPS attack
Primary Application Processor	CVE-2013-7266 CWE-20 CWE-789 CWE-770 CAPEC-130	Does not ensure length values match size of data structure Improper input validation Uncontrolled memory allocation Allocation of resources without limits or throttling Excessive allocation
I2C & RS-232 Protocols	CAPEC-272 CAPEC-220	Protocol manipulation Client-server protocol manipulation
Imagery Application Processor	CWE-805 CAPEC-100	Buffer access with incorrect length value Overflow buffers
Safety Switch Processor	CWE-1037	Processor optimization removal or modification of security-critical code
Laptop	CAPEC-615 CAPEC-604	Evil twin Wi-Fi attack Wi-Fi jamming
Camera	CVE-2014-6434	Allows remote attackers to execute commands in a restart action

ciate to the model to find the most relevant and strong evidence (Table I). This evidence is what ultimately informs other stakeholders, such that they can devise mitigative actions—changing the system solution to conform to mission needs, erecting security barriers at strategic points, or applying resilience solutions during operation.

## VII. RELATED WORK

Little research has been done for evidence-based security assessment in a model-based setting. Usually work in this area requires *transcribing* already known vulnerabilities to a modeling tool and assessing if it might apply to a system design. Instead, the aim of this work is to employ models that can—by their fidelity—immediately produce a large number of potential attack vectors. These attack vectors stem from the model itself and are not informed from some a priori security knowledge.

We acknowledge that some current attack vector search tools could be repurposed for model-based systems engineering. One such search tool is *cve-search* [29]. However, *cve-search* cannot input a system model. It only provisions security datasets in one search engine. It is also limited with respect to visualization techniques.

Noel et al. [30] propose CyGraph which also is based on a graph-based understanding of the system but this work fundamentally differs in scope (mainly targets traditional networked systems) and approach (uses a traditional notion of attack graphs).

Adams et al. [31] propose topic modeling for finding applicable attack vectors given a system model. However, they only examine CAPEC as a potential source of attack vectors, which is necessary but insufficient.

Ford et al. [32] propose using the ADVISE security methodology [33] on top of the Möbius tool [34] to provide an attacker’s view. However, the quantitative analysis is based

on profiling and modeling attacker actions. The framework is largely unaware of a specific system model that could be used to implement a realized system.

The analysis presented in this paper is qualitative. This is because quantitative information for cyber-physical attacks is limited and ultimately expert input is necessary to understand what it means for a metric to show that a system is more susceptible to attacks over another. For example, a number of quantitative approaches incorporate CVSS as a potential metric for risk [35, 36, 37]. But, CVSS only defines severity of a given vulnerability and not risk [38, 39].

In general, to the best of the authors’ knowledge, there is no direct comparison between the work in this paper and existing work in the literature. It is challenging to do a direct comparison with any existing models because previous work is based on an already implemented system or does not apply attack vector information directly to the model.

## VIII. CONCLUSION

In this paper we propose a method and implement a tool to support this method, CYBOK, that is able to find associated attack vectors given a sufficient system model. CYBOK provides flexibility in modifying the system model to represent different design solutions that implement the same desired behaviors. Therefore, moving security analysis earlier in the systems lifecycle—particularly at the design phase—and, therefore, building systems with security by design. Two important metrics are used for assessing a systems security posture; the attack surface and exploit chains. The results of this method and toolkit is illustrated and evaluated using a UAS—an important area for secure system design because exploits can cause hazardous behavior.

As a final observation we note the experience of using a systematic, model-driven process to conduct attack vector analysis often yields more information than just quantifying

the vulnerability aspects of the system. The process itself is an iterative learning experience, allowing circumspection into how a system behaves in response to potential exploits.

## IX. ACKNOWLEDGMENTS

This material is based upon work supported in part by the Center for Complex Systems and Enterprises at the Stevens Institute of Technology and in part by the United States Department of Defense through the Systems Engineering Research Center (SERC) under Contract HQ0034-13-D-0004. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Department of Defense.

## REFERENCES

- [1] E. Fong and V. Okun, "Web application scanners: Definitions and functions," in *Proceedings of the 40th Hawaii International International Conference on Systems Science (HICSS-40 2007)*, 2007.
- [2] S. Wisseman, "Third-party libraries are one of the most insecure parts of an application." Available from TechBeacon, <https://techbeacon.com/security/third-party-libraries-are-one-most-insecure-parts-application>.
- [3] P. Bisht, M. Heim, M. Iffland, M. Scovetta, and T. Skinner, "Managing security risks inherent in the use of third-party components," SAFECODE, Tech. Rep., 2017.
- [4] F. Frola and C. Miller, "System safety in aircraft acquisition," *Logistics Management Institute*, 1984.
- [5] A. Strafaci, "What does BIM mean for civil engineers," *CE News, Transportation*, 2008.
- [6] P. H. Nguyen, S. Ali, and T. Yue, "Model-based security engineering for cyber-physical systems: A systematic mapping study," *Information & Software Technology*, 2017.
- [7] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-based evaluation: From dependability to security," *Transactions on Dependable and Secure Computing*, 2004.
- [8] B. Chapman, "What are your intentions?" ;login:, 2001. [Online]. Available: <https://www.usenix.org/publications/login/july-2001-volume-26-number-4/what-are-your-intentions>
- [9] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, "Analysis of safety-critical computer failures in medical devices," *IEEE Security & Privacy*, 2013.
- [10] N. Kshetri and J. M. Voas, "Hacking power grids: A current problem," *Computer*, 2017.
- [11] G. Bakirtzis, B. T. Carter, C. R. Elks, and C. H. Fleming, "A model-based approach to security analysis for cyber-physical systems," in *Proceedings of the 2018 Annual IEEE International Systems Conference, SysCon 2018*, 2018.
- [12] J. Lambert, "Defenders think in lists. Attackers think in graphs. As long as this is true, attackers win." <https://perma.cc/6NZ2-A2HY>, 2015.
- [13] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, 2011.
- [14] B. T. Carter, G. Bakirtzis, C. R. Elks, and C. H. Fleming, "A systems approach for eliciting mission-centric security requirements," in *2018 Annual IEEE International Systems Conference (SysCon 2018)*, 2018.
- [15] R. Oates, F. Thom, and G. Herries, "Security-aware, model-based systems engineering with SysML," in *Proceedings of the 1st International Symposium on ICS & SCADA Cyber Security Research*, 2013.
- [16] U. Brandes, M. Eiglsperger, J. Lerner, and C. Pich, *Graph markup language (GraphML)*. CRC Press, 2013.
- [17] "Common Vulnerabilities and Exposures (CVE)," Available from MITRE, <https://cve.mitre.org/>.
- [18] "National Vulnerability Database (NVD)," Available from NIST, <https://nvd.nist.gov/>.
- [19] "Common Weakness Enumeration (CWE)," Available from MITRE, <https://cwe.mitre.org/>.
- [20] "Common Attack Pattern Enumeration and Classification (CAPEC)," Available from MITRE, <https://capec.mitre.org/>.
- [21] "Common Platform Enumeration (CPE)," Available from MITRE, <https://cpe.mitre.org/>.
- [22] "Exploit database (exploit-db)," Available from Offensive Security, <https://www.exploit-db.com/>.
- [23] G. Bakirtzis, B. J. Simon, C. H. Fleming, and C. R. Elks, "Looking for a black cat in a dark room: Security visualization for cyber-physical system design and analysis," *arXiv preprint arXiv:1808.08081*, 2018.
- [24] J. Jacobs and B. Rudis, *Data-driven security: analysis, visualization and dashboards*. John Wiley & Sons, 2014.
- [25] P. Beling, B. Horowitz, C. Fleming, S. Adams, G. Bakirtzis, B. Carter, T. Sherburne, C. Elks, A. Collins, and B. Simon, "Model-based engineering for functional risk assessment and design of cyber resilient systems," Systems Engineering Research Center (SERC), Tech. Rep., 2019.
- [26] G. L. Ward, G. Bakirtzis, and R. H. Klenke, "A modular software platform for unmanned aerial vehicle autopilot systems," in *52nd Aerospace Sciences Meeting*, ser. AIAA SciTech. American Institute of Aeronautics and Astronautics, Jan. 2014.
- [27] G. Bakirtzis, "bakirtzis/cybok-cli: first release," 2018. [Online]. Available: <https://doi.org/10.5281/zenodo.1313696>
- [28] G. Bakirtzis and B. J. Simon, "bakirtzis/graphml\_export: first release," 2018. [Online]. Available: <https://doi.org/10.5281/zenodo.1308914>
- [29] "cve-search," Available from CIRCL, <https://cve.circl.lu/>.
- [30] S. Noel, E. Harley, K. Tam, M. Limiero, and M. Share, "CyGraph: graph-based analytics and visualization for cybersecurity," in *Handbook of Statistics*. Elsevier, 2016, vol. 35.
- [31] S. Adams, B. Carter, C. Fleming, and P. A. Beling, "Selecting system specific cybersecurity attack patterns using topic modeling," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018.
- [32] M. D. Ford, K. Keefe, E. LeMay, W. H. Sanders, and C. Muehrcke, "Implementing the advise security modeling formalism in möbius," in *Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013.
- [33] E. LeMay, M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehrcke, "Model-based security metrics using adversary view security evaluation (advise)," in *Proceedings of the 2011 Eighth International Conference on Quantitative evaluation of systems (QEST)*. IEEE, 2011.
- [34] T. Courtney, S. Gaonkar, M. Griffith, V. Lam, M. McQuinn, E. Rozier, and W. H. Sanders, "Data analysis and visualization within the möbius modeling environment," in *Proceedings of the 2006 Third International Conference on Quantitative Evaluation of Systems (QEST)*. IEEE, 2006.
- [35] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring network security using dynamic bayesian network," in *Proceedings of the 4th ACM workshop on Quality of Protection*. ACM, 2008.
- [36] S. H. Houmb, V. N. Franqueira, and E. A. Engum, "Quantifying security risk level from CVSS estimates of frequency and impact," *Journal of Systems and Software*, 2010.
- [37] P. Wang, A. Ali, and W. Kelly, "Data security and threat modeling for smart city infrastructure," in *Cyber Security of Smart Cities, Industrial Control System and Communications (SSIC), 2015 International Conference on*. IEEE, 2015.
- [38] J. Spring, A. H. E. Hatleback, A. Manion, and D. Shic, "Towards improving CVSS," Software Engineering Institute, Carnegie Mellon University, Tech. Rep., December 2018.
- [39] Z. A. Collier, D. DiMase, S. Walters, M. M. Tehranipoor, J. H. Lambert, and I. Linkov, "Cybersecurity standards: Managing risk and creating resilience," *Computer*, 2014.