processes

MDPI

Article

Automatic Hybrid Attack Graph (AHAG) Generation for Complex Engineering Systems †

Mariam Ibrahim [1,*] and Ahmad Alsheikh [1,2]

[1] Department of Mechatronics Engineering, Faculty of Applied Technical Science, German Jordanian University, Amman 11180, Jordan; a.alsheikh@gju.edu.jo
[2] Department of Mechanical Engineering and Mechatronics, Deggendorf Institute of Technology, 94469 Deggendorf, Germany
* Correspondence: mariam.wajdi@gju.edu.jo
† This paper is an extended version of paper published in the international conference: 2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Iasi, Romania, 28–30 June 2018.

Received: 16 September 2019; Accepted: 21 October 2019; Published: 1 November 2019

Abstract: Complex Engineering Systems are subject to cyber-attacks due to inherited vulnerabilities in the underlying entities constituting them. System Resiliency is determined by its ability to return to a normal state under attacks. In order to analyze the resiliency under various attacks compromising the system, a new concept of Hybrid Attack Graph (HAG) is introduced. A HAG is a graph that captures the evolution of both logical and real values of system parameters under attack and recovery actions. The HAG is generated automatically and visualized using Java based tools. The results are illustrated through a communication network example.

Keywords: Hybrid Attack Graph; Level-of-Resilience; stability; topology

1. Introduction

As a result of the rapid advancement of complex engineering systems such as infrastructure, communications, energy systems, industrial automation, artificial intelligence, and cyber-physical systems, new research directions in modeling, monitoring, diagnosis, optimization, and control have emerged in recent years [1]. For instance, a control chart scheme for production processes was developed by [2] for monitoring the mean time between two events under the neutrosophic statistics using the belief estimator for the neutrosophic gamma distribution. The diagnosis and correction of many production problems which often cause huge loss to the production unit can substantially be improved with the utilization of the effective control chart technique.

In [3], the advantages of using Proportional-Integral (PI) controller for pH control in the raceway reactor during the whole day against traditional On/Off control were demonstrated. The paper also presented an event-based control architecture for Proportional–Integral–Derivative (PID) controllers. The objective is to tune a classical time-driven PI for pH control in the raceway reactor, and then to add event-based capabilities, but keeping the initial PI control design. The event-based systems allow a trade-off between control performance and control effort, which is perfect for the microalgae process in raceway reactors. The performed tests were oriented to establish a trade-off between control effort and control performance and present an alternative to traditional control.

The applicability of the Distributed Model Predictive Control (DiMPC) was investigated in [4] to deal with the constraints in the steam/water loop of a steam power plant. A comparison was conducted between the Decentralized Model Predictive Control (DeMPC), the Centralized Model Predictive Control (CMPC), and the DiMPC. The results showed the effectiveness of the DiMPC [5] designed an Optimal Nonlinear Adaptive Control (ONAC) strategy to achieve optimal parameter

tuning of Nonlinear Adaptive Control (NAC) for Voltage Source Converter (VSC) operating in both rectifier mode and inverter mode where an optimal and robust control can be achieved under different operation scenarios.

A novel pole-zero cancelation method was proposed by [6] for Multi-Input Multi-Output (MIMO) temperature control in heating process systems. In the proposed method, the temperature differences and the transient response of each point can be controlled by considering the dead time and the coupling effect of the MIMO system. In [7], the Slow-Mode-Based Control (SMBC) method combined with decoupling and dead-time compensation was applied to the MIMO temperature control system. The temperature differences and the transient response of all points can be controlled and improved by making the output of the fast modes follow that of the slow mode. The results were then compared to the conventional PI control and gradient temperature control methods.

The Deep Deterministic Policy Gradient (DDPG) technique for the optimum boost control on a Variable Geometry Turbocharger (VGT)-equipped engine is implemented by [8]. The proposed DDPG algorithm is compared with a fine-tuned PID controller to validate its optimality. The results showed that the control performance based on the proposed DDPG algorithm can achieve a good transient control performance from scratch by autonomously learning the interaction with the environment, without relying on model supervision or complete environment models.

A framework was proposed by [9] to use the structural information in each Possible Conflict (PC) for fault diagnosis of complex industrial system to design a different kind of executable model. They proposed to build grey-box models based on a state space neural network architecture derived from that structural information in the PC, which links measurements with equations, and consequently with parameters related to faulty behavior. The state space Neural Networks (ssNN) were used to track the system behavior, but once a fault detection was confirmed, the structural information in the models and the consistency-based diagnosis paradigm were used to perform fault isolation.

Total decomposition of nonstationary variables for distributed monitoring of nonstationary industrial processes was handled by [10], in which different variable blocks were separated with both overlapping and nonoverlapping relationships considered, capturing different nonstationary characteristics. A two-level monitoring strategy was designed that can supervise both the local cointegration relationships and the interrelationship among different nonstationary blocks with enhanced interpretation of the nonstationary process.

A novel diagnosis framework was proposed by [11] for considering the deep feature learning and cross-domain feature distribution alignment simultaneously for industrial applications. Extending the Marginal Distribution Adaptation (MDA) to Joint Distribution Adaptation (JDA), the proposed framework can exploit the discrimination structures associated with the labeled data in source domain to adapt the conditional distribution of unlabeled target data, and thus, guarantee a more accurate distribution matching [12]. They reviewed over 220 technical research programs in total, with more attention on the recent developments of the fault diagnosis approaches and their applications during the last decade. Knowledge-based fault diagnosis, hybrid fault diagnosis, and active fault diagnosis were reviewed comprehensively. The distinctive advantages and various constrains of these diagnosis methods were commented on. A recent survey by [1] also summarized papers on monitoring and diagnosis for complex engineering systems.

The implementation of diagnostic and prognostic architectures can aid the implementation of advanced control algorithms in a resilient control system to recognize sensor degradation, as well as failures with industrial process equipment associated with the control algorithms [13]. A resilient control system is one that maintains state awareness and an accepted level of operational normalcy in response to disturbances, including threats of an unexpected and malicious nature [14]. As a result, it is hard to obtain true expectations about the consequences when a fault/attack occurs effecting or compromising the system. Hence, evaluating systems resilience in terms of stability, performance, and recovery time is crucial and valuable for cost management and design tradeoff. The traditional Attack graphs can generate various attack scenarios compromising the system in terms of violations

of a security property. However, they are only concerned with tracking the logical changes in the system parameters under attacks, as captured by the pre and post-conditions. Therefore, the novelty of this work lies in introducing and generating automatically a Hybrid Attack Graph (HAG) using our new Automatic Hybrid Attack Graph (AHAG) Java based tool that combines logical and real values of system parameters. In fact, HAG can be used to provide additional information about the expected changes that could affect the system state by different attack scenarios constituting the graph. In such cases, attack scenarios could be compared for the worst attack scenario that would compromise a system. This can be done by determining the associated real values (i.e., the resilience levels as determined in [15]). The results are illustrated through two-communication networks example. The networks models and security properties (written using Architecture Analysis & Design Language (AADL) [16] and AADL Annex Assume Guarantee REasoning Environment (AGREE) plug-in [17] that relies on JKind model-checker tool [18]) are fed to the AHAG tool, which generates all possible attack scenarios of the systems model and visualizes the graphs using Unity software [19].

This paper extends the conference version [20] significantly by introducing the concept of the Hybrid Attack Graph (HAG), and a new developed tool for its automatic generation (AHAG). The tool is implemented on communication networks example. The remainder of this paper is organized as follows: Section 1.1 reviews the related work. Section 2 describes the model based attack graph implementation through illustrative communication networks example. Section 3 explains the Level-of-resilience assessment. Section 4 presents the Hybrid Attack Graph. Section 5 introduces our AHAG tool to automatically generate the Hybrid attack graph and shows the experimental results. Section 6 summarizes and presents certain future directions.

1.1. Related Work

Existing Attack graph generation tools can be summarized as follows [21]. One research implemented a tool that consisted of three main pieces: a model builder (it takes as input information about network topology, configuration, and a library of attack rules), an attack graph generator (SPIN), and a Graphical User Interface (GUI) for graphical presentation. Topological Vulnerability Analysis (TVA), Network Security Planning Architecture (NETSPA), and Multi-host Multi-stage Vulnerability Analysis (MULVAL) tools [22]. These tools can explore all potential methods an attacker can use to corrupt an enterprise network by determining the configuration information of the hosts and the network. The Cauldron tool implemented by [23] automatically mapped all paths of vulnerability by correlating, aggregating, normalizing, and fusing data from various assets. The Network Attack Graph GENeration (Naggen) tool is developed by [24] to generate attack graph. Other research [5] has illustrated how logical attack graph complexity can be simply elevated when a network becomes denser and larger [25]. They utilized the MULVAL tool, but they changed its engine so that the trace of evaluation was recorded and sent to a graph builder.

In [26], the New Symbolic Model Checker (NuSMV) model checker was implemented to develop Attack graph counterexamples. The value iteration method was also implemented to determine the reliability of Attack graph, which allowed designers to identify which nominal set of security defenses would assure systems safety. A model-checking-based Automated Attack Graph Generator and Visualizer (A2G2V) was proposed by [27]. The proposed A2G2V algorithm used existing model-checking tools, an architecture description tool, and C code to generate an attack graph that enumerates the set of all possible sequences in which atomic-level vulnerabilities can be exploited to compromise system security. The A2G2V tool required building three main functions: a counterexample parsing function, a cyclic testing function, and a Luster model editing function.

In [28], a heuristic algorithm was used to automatically determine the optimal security hardening through cost-benefit analysis [29]. They used multiple algorithms, such as Vulnerability Node Matching, Attack Graph Optimization, Maximum Loss Flow, Path Seeking, and Multi-Objective Augmented Road Sorting, to investigate the vulnerability, estimate the global path, and determine the optimal attack path. The surveys conducted by [30,31] illustrated the state-of-the-art technologies in Attack graph construction in computer networks, and their potential development challenges. Alerts were also analyzed from intrusion detection system, and how well the approaches scaled to larger networks.

A Hybrid Attack Graph (HAG) was modeled by [32], using linguistics and type extensions of traditional attack graph given the necessary inputs of asset, topology, and fact declarations in the typed grammar. The HAG was illustrated through an example of an attacker who is trying to compromise an automotive car by forcing it to drivetoward a wall. The tool automates the task of creating HAGs by compiling the inputs specified and making the appropriate connections. A Hybrid Attack Dependency Graph (HADG) was presented by [33], which allowed discretization into intervals of the reachable and related ranges of the system's state variables and their evolution over the execution of attacks with duration. The HAG generation software of [33] was used by [34] to model a Cyber physical System's attack for a smart grid in which an attacker has to obtain access to a Supervisory Control and Data Acquisition (SCADA) system to cause the transformer to overheat. Thus, transformer temperature was the continuous value in question and was discretized into intervals. The HAG was built by matching exploit patterns to a particular system state.

The concept of adding priorities to HAGs was introduced by [35]; if multiple exploit preconditions were met in a given state, only those with the highest priority value would be expanded into new states of the attack graph. Lower priority exploits would then only be considered if all exploits with higher priorities failed to meet their preconditions. By applying exploit priorities as a heuristic measure, attack graph states can be explored in a more strategic manner.

A novel Hybrid Attack Model (HAM) was introduced by [36] that combines Probabilistic Learning Attacker, Dynamic Defender (PLADD) game model and a Markov Chain model to simulate the planning and execution stages of a bad data injection attack in power grid. The hybrid model is shown to be capable of modeling long time-to-completion actions in the preparation stage and short time-to-completion actions in the execution stage. Table 1 summarizes the main characteristics and limitations of the existed studies in HAG generation as compared to our Automatic Hybrid Attack Graph (AHAG) tool.

Table 1. Main characteristics and limitations of the existed studies in Hybrid Attack Graph (HAG) generation.

	Main Characteristics	**Limitations**
HAG [32]	• Stateful graph. • Captures blended attack vectors (comprising discrete and continuous exploit events). • Time is implemented using a single group of global exploits, each of which increments a class of assets' position in time depending upon its particular state.	• Generation of attack graphs from nontrivial scenarios (with tens of hybrid assets) in acceptable time is daunting.

Table 1. *Cont.*

	Main Characteristics	Limitations
Hybrid Attack Dependency Graph (HADG) [33]	• Replaces the state transition graph with a dependency graph. • A hybrid attack takes a range of real values as preconditions and output a range of values as postconditions. • Provides the capability of modeling continuous state variables and their evolution over the execution of attacks with duration.	• The process for generating HADGs must be articulated and formalized and its performance characterized to better handle continuous variables. • Expansion of the exploit list and the asset parameters in the network file is required to fully explore the attack space.
HAG [36]	• Hybrid Attack Model (HAM) consists of both Probabilistic Learning Attacker, Dynamic Defender (PLADD) nodes and Markov state nodes. • The attacker must have control over all PLADD nodes, which represent the preparation for an attack, to be able to execute an attack by traversing through the Markov states. • Considers the time difference between the attacker's action to prepare for an attack in comparison to the attacker's action to execute for an attack.	• The number of time steps in the Markov Chain model per unit time in the PLADD model must be specified by a domain expert. • The PLADD and the Markov Chain models do not have the means to run a simulation where the time-to-completion of an action for attacker and defender can be significantly different in the preparation stage and the execution stage.
Automatic Hybrid Attack Graph (AHAG)	• Stateful graph. • Specifies attack patterns as sets of preconditions and postcondition. • The integration of level of resilience values captures the system response under sequential attack and recovery actions.	• Requires comprehensive overview of the system model and security/resiliency property. • Levels-of-Resilience values need to be determined under attacks.

2. Model-Based Attack Graph Implementation

2.1. Networked Systems Examples

The two images of Figure 1a,b show two communication networks with identical clients and services (Email, File Transfer Protocol (FTP), and Video), but with different topologies [15]. In both networks, the routers R_1, R_2, and R_3 are built with Routing Protocol (RIP) [37], where the traffic is rerouted under faults as a recovery action through a redundant path (if it exists) based on hop count [37]. In addition to that, R_1 is linked to three Local Area Networks (*LANs*); LAN_1, LAN_2, and LAN_3, where every *LAN* has 10 clients. R_2 is linked to LAN_4, which has 10 clients as well. Router R_3 has three more links, which connect it, respectively, to an Email server through the Internet, an FTP server, and a Video workstation. The second communication network, CN_2, has similar clients and services as CN_1, but has a different topology.

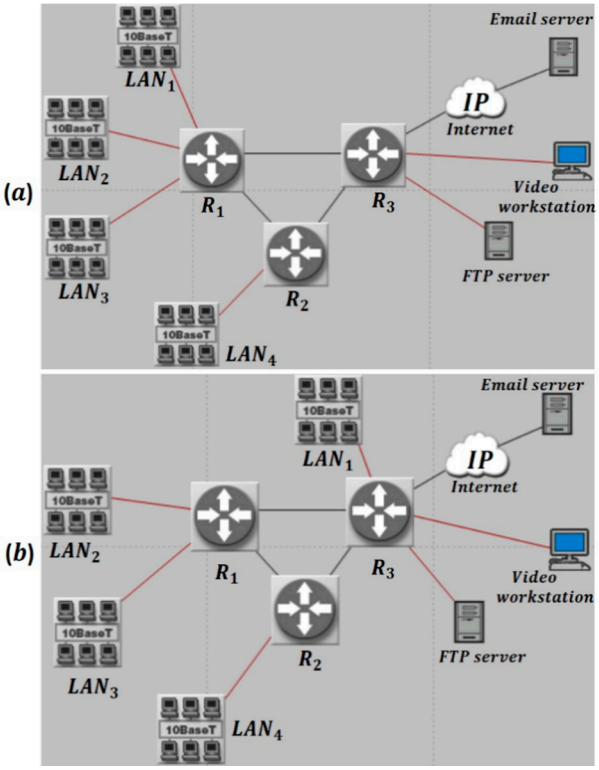

Figure 1. (**a**) Network CN_1 and (**b**) Network CN_2.

2.2. Formal System Description for Networked Examples

The generation of Attack Graph requires an overall formal description of the system model and security property being investigated, encoded in Architecture Analysis and Design Language (AADL) and checked using JKind. Here, the formal descriptions of CN_1 and CN_2, respectively, are given as follows [20].

1. Set of Routers R = 1, 2, 3, IP Cloud; Variable $I \in \{1, 2, 3, 4\}$ (static parameters).
2. Set of LANs N = 1, 2, 3, 4; Variable $k \in \{1, 2, 3, 4\}$ (static parameters).
3. Set of Service Providers S; Variable $s \in$ {Ftp, Email, Video} (static parameters).
4. Set of Connection Links $L \subseteq R \times R$, $R \times N$, $R \times S$; Labeled $l_{ij} \equiv$ Link is placed between component i and component j (static parameters).
5. System Connectivity C = L; Boolean $c_{ij} = 1$ if there is a connection between component i and component j (dynamic variables).
6. System Stability T; Boolean t = 1 if system is stable (dynamic variable).
7. System Performance $P \subseteq S$; Boolean $f_k = 1$ if ftp service is provided to LAN k, Boolean $e_k = 1$ if Email service is provided on LAN k and Boolean $v_k = 1$ if Video service is provided on LAN k (dynamic variables).
8. System recovery Action R; Variable $r \in$ {p, a, d}, in case of normal operation r = p, in case of recovery action r = a, and in case of no action can be done, r = d (dynamic variables).
9. Number of faulted Links that occur sequentially N; Variable $n \in \{0, 1, 2\}$, in case of no fault $n = 0$, in case of first fault $n = 1$, and in case of second fault $n = 2$ (dynamic variables).
10. Attack Instance $AI \subseteq A \times R \times R$, $A \times R \times N$, $A \times R \times S$, Labeled $a_{ij}{}^{m} \equiv$ Attack a on the Link between component i and component j, where $m \subseteq L$ is a sequence of the previous faulted link(s) if exists. (static parameters)
11. Pre-Attack conditions for CN_1:

- $Pre(a_{13}) \equiv (c_{13} = 1) \wedge (t = 1) \wedge (r = p) \wedge (n = 0)$
- $Pre(a_{12}) \equiv (c_{12} = 1) \wedge (t = 1) \wedge (r = p) \wedge (n = 0)$
- $Pre(a_{23}) \equiv (c_{23} = 1) \wedge (t = 1) \wedge (r = p) \wedge (n = 0)$
- $Pre(a_{23}^{13}) \equiv (c_{23} = 1) \wedge (r = a) \wedge (n = 1)$
- $Pre(a_{23}^{12}) \equiv (c_{23} = 1) \wedge (r = p) \wedge (n = 1)$
- $Pre(a_{13}^{23}) \equiv (c_{13} = 1) \wedge (r = a) \wedge (n = 1)$
- $Pre(a_{13}^{12}) \equiv (c_{13} = 1) \wedge (r = p) \wedge (n = 1)$
- $Pre(a_{12}^{13}) \equiv (c_{12} = 1) \wedge (r = a) \wedge (n = 1)$
- $Pre(a_{12}^{23}) \equiv (c_{12} = 1) \wedge (r = a) \wedge (n = 1)$.

12. Post-Attack conditions for CN_1:

- $Post(a_{13}) \equiv (c_{13} = 0) \wedge (r = a) \wedge (n = 1)$
- $Post(a_{12}) \equiv (c_{12} = 0) \wedge (r = p) \wedge (n = 1)$
- $Post(a_{23}) \equiv (c_{23} = 0) \wedge (r = a) \wedge (n = 1)$
- $Post(a_{23}^{13}) \equiv (t = 0) \wedge (c_{23} = 0) \wedge (f_1 = f_2 = f_3 = f_4 = e_1 = e_3 = e_4 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{23}^{12}) \equiv (t = 0) \wedge (c_{23} = 0) \wedge (f_4 = e_4 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{13}^{23}) \equiv (t = 0) \wedge (c_{13} = 0) \wedge (f_1 = f_2 = f_3 = f_4 = e_1 = e_3 = e_4 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{13}^{12}) \equiv (t = 0) \wedge (c_{13} = 0) \wedge (f_1 = f_2 = f_3 = e_1 = e_3 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{12}^{13}) \equiv (t = 0) \wedge (c_{12} = 0) \wedge (f_1 = f_2 = f_3 = e_1 = e_3 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{12}^{23}) \equiv (t = 0) \wedge (c_{12} = 0) \wedge (f_4 = e_4 = 0) \wedge (r = d) \wedge (n = 2)$

13. Pre-Attack conditions for CN_2:

- $Pre(a_{13}) \equiv (c_{13} = 1) \wedge (t = 1) \wedge (r = p) \wedge (n = 0)$
- $Pre(a_{12}) \equiv (c_{12} = 1) \wedge (t = 1) \wedge (r = p) \wedge (n = 0)$
- $Pre(a_{23}) \equiv (c_{23} = 1) \wedge (t = 1) \wedge (r = p) \wedge (n = 0)$
- $Pre(a_{23}^{13}) \equiv (c_{23} = 1) \wedge (r = a) \wedge (n = 1)$
- $Pre(a_{23}^{12}) \equiv (c_{23} = 1) \wedge (r = p) \wedge (n = 1)$
- $Pre(a_{13}^{23}) \equiv (c_{13} = 1) \wedge (r = a) \wedge (n = 1)$
- $Pre(a_{13}^{12}) \equiv (c_{13} = 1) \wedge (r = p) \wedge (n = 1)$
- $Pre(a_{12}^{13}) \equiv (c_{12} = 1) \wedge (r = a) \wedge (n = 1)$
- $Pre(a_{12}^{23}) \equiv (c_{12} = 1) \wedge (r = a) \wedge (n = 1)$.

14. Post-Attack conditions for CN_2:

- $Post(a_{13}) \equiv (c_{13} = 0) \wedge (r = a) \wedge (n = 1)$
- $Post(a_{12}) \equiv (c_{12} = 0) \wedge (r = p) \wedge (n = 1)$
- $Post(a_{23}) \equiv (c_{23} = 0) \wedge (r = a) \wedge (n = 1)$
- $Post(a_{23}^{13}) \equiv (t = 0) \wedge (c_{23} = 0) \wedge (f_2 = f_3 = f_4 = e_3 = e_4 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{23}^{12}) \equiv (t = 0) \wedge (c_{23} = 0) \wedge (f_4 = e_4 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{13}^{23}) \equiv (t = 0) \wedge (c_{13} = 0) \wedge (f_2 = f_3 = f_4 = e_3 = e_4 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{13}^{12}) \equiv (t = 0) \wedge (c_{13} = 0) \wedge (f_2 = f_3 = e_3 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{12}^{13}) \equiv (t = 0) \wedge (c_{12} = 0) \wedge (f_2 = f_3 = e_3 = v_2 = v_3 = 0) \wedge (r = d) \wedge (n = 2)$
- $Post(a_{12}^{23}) \equiv (t = 0) \wedge (c_{12} = 0) \wedge (f_4 = e_4 = 0) \wedge (r = d) \wedge (n = 2)$

15. Initial state: $(t = 1) \wedge (c_{23} = c_{12} = c_{13} = 1) \wedge (f_1 = f_2 = f_3 = f_4 = e_1 = e_3 = e_4 = v_2 = v_3 = 1) \wedge (r = p) \wedge (n = 0)$. (Initially, the system is stable, normally operated, and no service outages).
16. The security/resiliency property φ is that both CN_1 and CN_2 are always stable under the given attacks/faults. This can then be written by a CTL formula: $\varphi \equiv AG(t = 1) \equiv AG(\neg (t = 0))$.

2.3. Attack Scenarios Implementation

In this section, we present the Attack Graphs resulted by running the JKind model checker for the encoded AADL CN_1 and CN_2 descriptions, respectively, against the security property φ [20]. JKind is an infinite state model checker for checking safety properties of synchronous systems [38], which are expressed in Lustre, a formally defined, declarative, and synchronous dataflow programming language for programming reactive systems [39]. The Verification is based on k-induction and property directed reachability using a back-end Satisfiability Modulo Theories (SMT) solver. A verified property is determined to be true for all runs of the system. A property violation is reported with an explicit Counter-Example (CE), which is given here as an attack scenario (a sequence of attack and recovery actions resulting in system disruption).

In our work, the CN_1 and CN_2 descriptive models included entities and their interfaces and connections, which were encoded using Architecture Analysis and Design Language (AADL), within the open-source integrated development environment (Osate2). The *AADL* models were embedded with the AGREE Annex plug-in [17] that is used to specify the component models and system-level security properties. AGREE also translates the AADL+Annex models and properties to Lustre language, which JKind can verify against a security property of concern and delivers the result as a CE, if it exists [40].

Figures 2 and 3 show the Attack graphs for CN_1 and CN_2, respectively [20], capturing the state evolution of the two networks dynamical variables given in the earlier formal description under attack instances. These graphs are visualized using *Unity* tool that supports two-dimensional (2D) and three-dimensional (3D) graphics, drag-and-drop functionality, and scripting using C# [41].

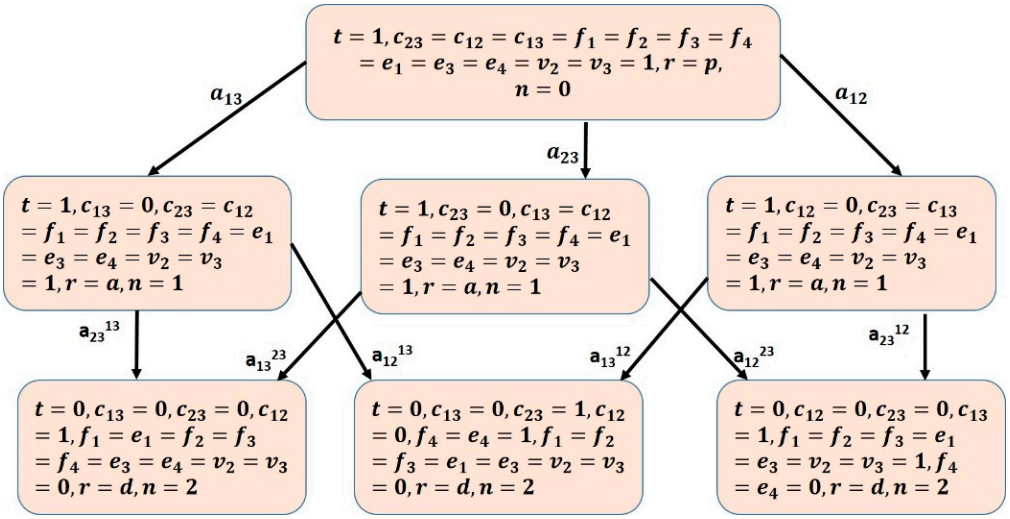

Figure 2. Network CN_1 Attack Graph.

From the obtained graphs it can be seen that both networks have six attack scenarios, resulting in networks loss of stability as determined by the unbounded traffic loss over time [42]. Each attack is a sequence of faults and recovery actions occurring sequentially as follow:

S_1: $a_{13} \rightarrow a_{23}{}^{13}$
S_2: $a_{13} \rightarrow a_{12}{}^{13}$
S_3: $a_{12} \rightarrow a_{23}{}^{12}$
S_4: $a_{23} \rightarrow a_{13}{}^{23}$
S_5: $a_{23} \rightarrow a_{12}{}^{23}$
S_6: $a_{12} \rightarrow a_{13}{}^{12}$

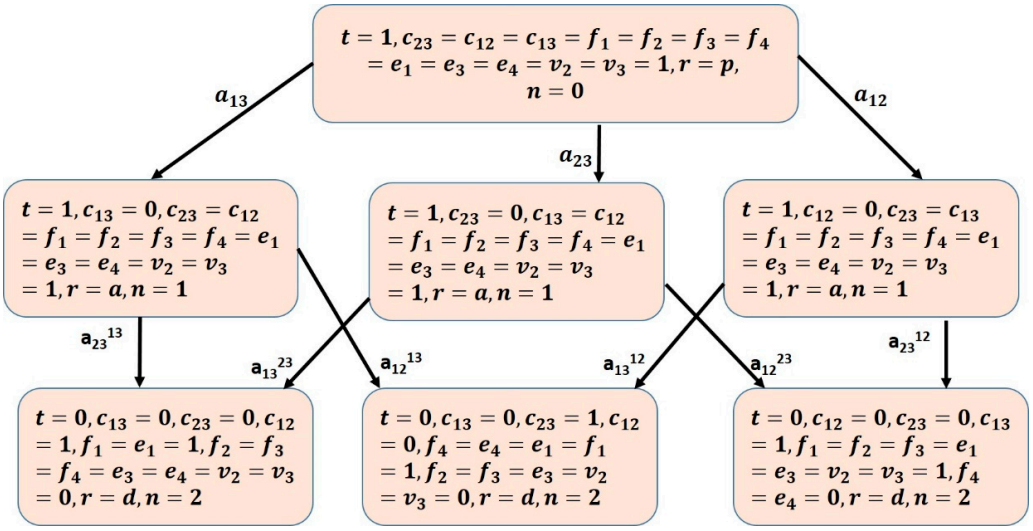

Figure 3. Network CN_2 Attack Graph.

3. Level-of-Resilience Assessment

Each path in the graph is a single attack scenario and has an associated Level-of-Resilience (LoR) [20]. The following definition can be utilized to identify the worst case Level-of-Resilience of a system with its Attack graph given. A system is the worst resilient to an attack scenario in the graph if it acquires the highest loss of stability, the highest loss of performance, or the highest recovery-time.

Definition 1 ([20]). *Given a system M and an attack graph A_G comprising a set of attack scenarios $S \equiv \cup S_i$, $i \in \{1, \ldots, z\}$, where z is the number of attack scenarios, we say that $LoR(M, S_i)$ is the worst if:*

$$\begin{gathered}[LoS_R(M, S_i) > LoS_R(M, S - S_i)] \\ \vee [[LoS_R(M, S_i) = LoS_R(M, S - S_i) \\ \wedge [LoP_R(M, S_i) > LoP_R(M, S - S_i)]] \\ \vee [[LoS_R(M, S_i) = LoS_R(M, S - S_i)] \\ \wedge [LoP_R(M, S_i) = LoP_R(M, S - S_i)] \\ \wedge [RT(M, S_i) > RT(M, S - S_i)]]\end{gathered}$$

The next definition compares the LoR of many systems against an Attack scenario. A system is the most resilient to an attack scenario if this attack acquires a smallest loss of stability, a smallest loss of performance, or smallest recovery-time.

Definition 2 ([20]). *Given a set of systems $M \equiv \cup M_j$, $j \in \{1, \ldots, y\}$, where y is the number of systems, and an attack scenario $S_i \in S$, we say that $LoR(M_i, S_i) > LoR(M\text{-}M_i, S_i)$ if:*

$$\begin{gathered}[LoS_R(M_i, S_i) < LoS_R(M - M_i, S_i)] \\ \vee [[LoS_R(M_i, S_i) = LoS_R(M - M_i, S_i)] \\ \wedge [LoP_R(M_i, S_i) < LoP_R(M - M_i, S_i)]] \\ \vee [[LoS_R(M_i, S_i) = LoS_R(M - M_i, S_i)] \\ \wedge [LoP_R(M_i, S_i) = LoP_R(M - M_i, S_i)] \\ \wedge [RT(M_i, S_i) < RT(M - M_i, S_i)]].\end{gathered}$$

4. Hybrid Attack Graph (HAG)

The states of Attack graph reflect the logical evolution of system parameters (e.g., system stability and services are either true or false) during attacks until reaching the final states where the security property φ is violated and hence are the worst states. As multiple attacks can reach the same final state, it is essential to identify the worst attack scenario. A Hybrid Attack Graph (HAG) associates real values to all attack scenarios terminating in these final states. These real values correspond to the Level-of-Resilience parameters determined from the system dynamical response. The detailed computations of LoS_R, eventual $\underline{\text{LoP}_\text{R}}$ and RT of networks CN_1 and CN_2 under A_G scenarios are given in [43], and [15]. We formally define HAG as follows.

Definition 3. *A Hybrid Attack Graph (HAG) of a system model M is a data structure representing a union of all attack paths comprising A_G annotated with the associated Levels-of-Resilience.*

$$HAG = A_G \cup LoR(M, A_G).$$

Algorithm 1 compares the Levels-of-Resilience, associated with the attacks comprising an Attack graph, and alerts the worst-case scenario.

Algorithm 1 Alerting Worst-Case Attack Scenario

INPUT: Attack graph (A_G) Comprising Attack Scenarios ($S \equiv \cup S_i, i \in \{1, \ldots, z\}$) and associated LoR values
OUTPUT: Alert Worst Case $S_{[i]}$
Procedure:
for Case [LoS_R, LoP_R, RT]
if $S_{[i]}\{\text{LoS}_\text{R}\} > S{-}S_{[i]}\ \{\text{LoS}_\text{R}\}$:
Alert Worst Case $S_{[i]}$
else if S[i] {LoP_R, Ftp} > $S{-}S_{[i]}$ {LoP_R, Ftp}
Alert Worst Case $S_{[i]}$
else if $S_{[i]}\{\text{LoP}_\text{R}$, Video} > $S{-}S_{[i]}$ {LoP_R, Video}
Alert Worst Case $S_{[i]}$
else if $S_{[i]}\{\text{LoP}_\text{R}$, Email} > $S{-}S_{[i]}\{\text{LoP}_\text{R}$, Email}
Alert Worst Case S[i]
else if $S_{[i]}\{\text{RT}\} > S{-}S_{[i]}\ \{\text{RT}\}$:
Alert Worst Case $S_{[i]}$
else
no Alert

5. Automatic Hybrid Attack Graph (AHAG) Tool

In Section 2, the attack scenarios were generated by repeatedly running AGREE, and updating the security property in every run to exclude the previously generated attack scenarios. Here, the automatic generation of attack scenarios is presented.

The Automatic Hybrid Attack Graph (AHAG) tool was developed through NetBeans, an Integrated Development Environment (IDE) for Java [44]. *NetBeans* allows applications to be developed from a set of modular software components called modules. In addition to Java development, NetBeans has extensions for other languages like Personal Home Page (PHP), *C*, C++, and fifth version of Hyper Text Markup Language (HTML5). Maven, a build automation tool used primarily for Java projects, was selected as the main project for the AHAG tool in NetBeans [45].

When running an AGREE based JKind model-checker for Lustre models within Osate2, it can only produce one counterexample at a time. However, if executed more than once, it may repeat the same counterexample. The AHAG tool shown in Figure 4 confirms that a different new counterexample is produced (if exists) each time the JKind is called, and automatically, all possible attack scenarios are visualized using Unity software.

AHAG tool takes only the first Lustre model (a translation of the system model and the security property from AGREE). Then, it generates all possible combinations of potential attack scenarios (i.e., CEs) as (.lus) format files. Next, AHAG communicates with the model checker JKind through the Command Prompt Commands (CMD) to iteratively check the system model and the potential CE against the security property within the Lustre files. Doing so, AHAG generates all corresponding results as separate Excel files, illustrating if the potential CE is truly an attack scenario or not. Once all Excel files are generated, AHAG converts them to (.csv) files to be used later along with LoR values in Unity Visualizer.

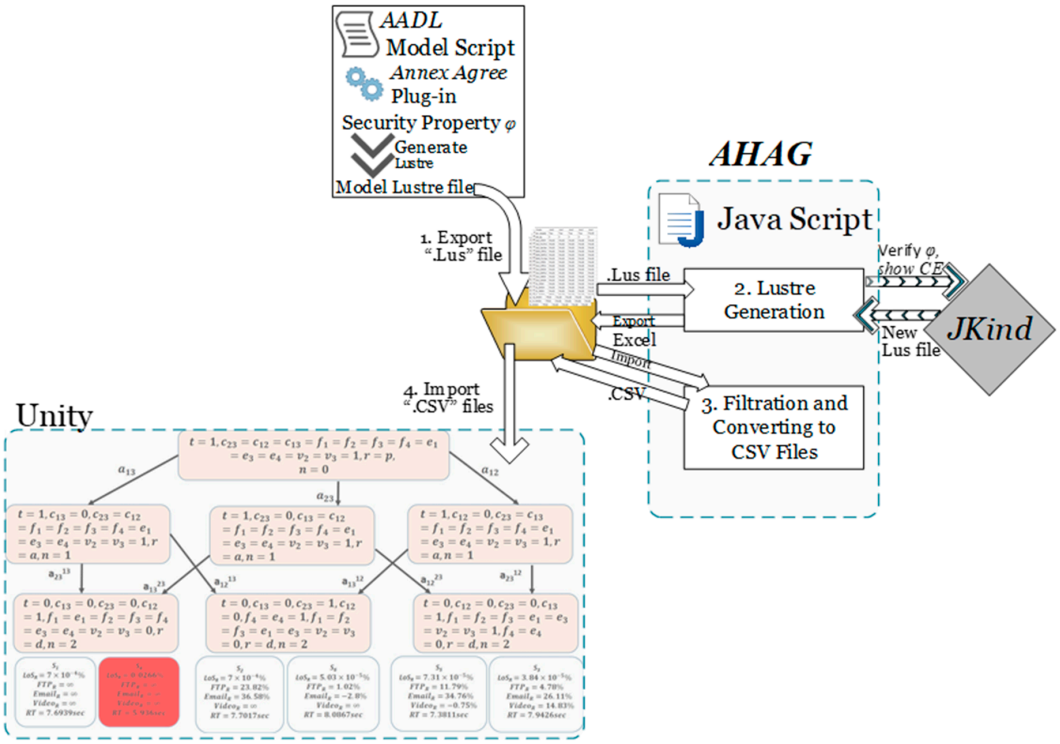

Figure 4. *AHAG* Workflow. AADL: Architecture Analysis & Design Language.

The AHAG Algorithm 2 is as follows. The user inserts the first Lustre Model as input and defines the attack instances/actions constituting the attack paths in a single-dimensional array. In addition to that, the user has to choose the maximum expected length n of an attack scenario. AHAG in turns, generates all potential combinations of attack scenarios A^n, where A is number of attack instances/actions. Each new potential attack scenario is stored as a variable CE_1 within (.lus) generated files. Next, JKind is called iteratively to check these files against the security property ϕ. If CE_1 violates the security property, then it is a true attack scenario, which belongs to the Attack graph, and the result is given as an Excel sheet (.xlsx). Otherwise, AHAG will reject the CE_1. Afterward, since it is easier for Unity visualizer to read (.csv) files, AHAG converts xlsx to csv files and feed them to Unity.

Algorithm 2 AHAG

INPUT: System Model (Luster 0.lus), Attack instances A[ ], maximum length (n)
OUTPUT: All attack scenario in (.csv)
Procedure:
insert attack instances in the Single-Dimensional Array (A)
set maximum length (n)
loop 1:
CE_1 = possible combination from A[ ] of length n
New Lustre = do new (.lus) copy (Lustre 0.lus)
New Lustre = New Lustre + CE_1
goto loop 1
loop 2:
call JKIND through cmd
New Lustre.xlsx= do result in (.xlsx) format
loop 3:
New Lustre.csv = replace (New Lustre.xlsx) format to (.csv) format
goto loop 3
goto loop 2
loop 4:
if New Lustre.csv contains CE_1 = False
delete (New Lustre.csv)
goto loop 4
generate violating attack scenarios

The generated attack scenarios were visualized using Unity visualizer tool and the Levels-of-Resilience values, including Level-of-Stability-Reduction (LoS_R), Level-of-Performance-Reduction (LoP_R) in the networks applications (given by FTP_R, $Email_R$, and $Video_R$), and the Recovery-Time (RT) are annotated to the attack scenarios terminating at the final states using Algorithm 1. This generates the HAG for CN_1 and CN_2, as shown in Figures 5 and 6, respectively. It can be seen that Attack scenario S_4 was chosen as the worst attack scenario (highlighted in red). The detailed computations of Levels-of-Resilience values for networks CN_1 and CN_2 are given in [43], and [15]. By comparing the generated HAGs to the traditional Attack graphs of Figures 2 and 3, respectively, it is clear how HAG can aid in differentiating attack scenarios terminating in the same state, yet with distinguishable resilience levels. Thus, network designers can have a better overview on the attack scenario that the network is most vulnerable to.

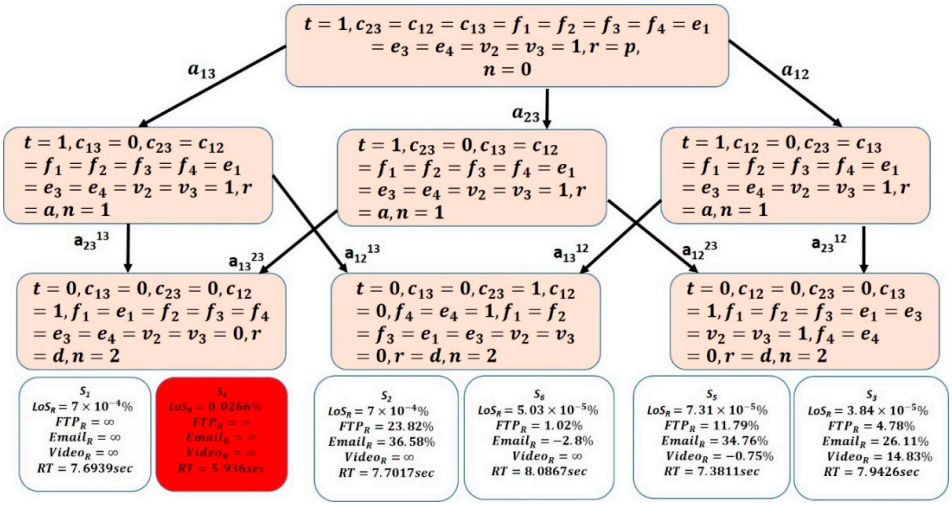

Figure 5. Network CN_1 Hybrid Attack Graph.

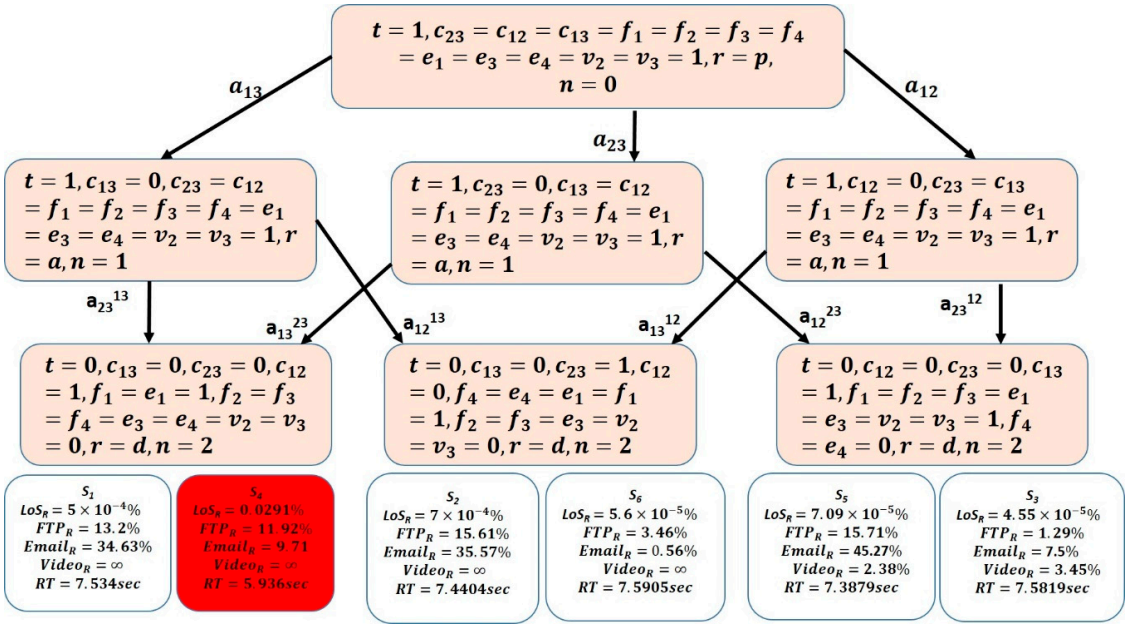

Figure 6. Network CN_2 Hybrid Attack Graph.

6. Conclusions

In this paper, we presented a new concept of Hybrid Attack Graph (HAG), which captures both the logical changes in the system parameters under attacks (as determined by the pre and post-conditions), and the real values of the levels of resilience parameters associated with the attacks constituting the graph. The nearest C-language based tool, Automated Attack Graph Generator and Visualizer (A2G2V), proposed by [27], interacts with the model-checker JKind for the generation of the attack paths one at a time, and with another tool Graph visualization (Graphviz) for visual display of the attack graph. Similar to our Automatic Hybrid Attack Graph (AHAG) tool, A2G2V tool requires one-time modeling effort to obtain the system description for components, connectivity, services, and their vulnerabilities. However, AHAG integrates the Levels-of-Resilience values obtained from system's dynamical response with the Attack graph. This automatically generates a HAG, which can aid system designers to investigate and select the worst LoR system design and its corresponding attack scenario from the graph. This ensures an appropriate defense and countermeasures placement in the system. The results were illustrated through a communication networks example.

Author Contributions: Conceptualization, M.I.; Methodology, M.I.; Software, M.I., A.A.; Validation, M.I., A.A.; Formal Analysis M.I.; Investigation, M.I., A.A.; Resources, M.I.; Data Curation, M.I., A.A.; Writing-Original Draft Preparation, M.I., A.A.; Writing-Review & Editing, M.I.; Visualization, M.I., A.A.; Supervision, M.I.; Project Administration, M.I.; Funding Acquisition, M.I.

Funding: This research was funded by Deanship of Graduation Studies and Scientific Research at the German Jordanian University for the seed fund SATS 02/2018.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gao, Z.; Nguang, S.K.; Kong, D.X. Advances in Modelling, monitoring, and control for complex industrial systems. *Complexity* **2019**, *2019*, 2975083. [CrossRef]
2. Aslam, M.; Bantan, R.A.R.; Khan, N. Monitoring the Process Based on Belief Statistic for Neutrosophic Gamma Distributed Product. *Processes* **2019**, *7*, 209. [CrossRef]
3. Rodríguez-Miranda, E.; Beschi, M.; Guzmán, J.L.; Berenguel, M.; Visioli, A. Daytime/Nighttime Event-Based PI Control for the pH of a Microalgae Raceway Reactor. *Processes* **2019**, *7*, 247. [CrossRef]

4. Zhao, S.; Maxim, A.; Liu, S.; De Keyser, R.; Ionescu, C.M. Distributed Model Predictive Control of Steam/Water Loop in Large Scale Ships. *Processes* **2019**, *7*, 442. [CrossRef]
5. Jiang, Y.; Jin, X.; Wang, H.; Fu, Y.; Ge, W.; Yang, B.; Yu, T. Optimal Nonlinear Adaptive Control for Voltage Source Converters via Memetic Salp Swarm Algorithm: Design and Hardware Implementation. *Processes* **2019**, *7*, 490. [CrossRef]
6. Xu, S.; Hashimoto, S.; Jiang, W. Pole-Zero Cancellation Method for Multi Input Multi Output (MIMO) Temperature Control in Heating Process System. *Processes* **2019**, *7*, 497. [CrossRef]
7. Xu, S.; Hashimoto, S.; Jiang, W.; Jiang, Y.; Izaki, K.; Kihara, T.; Ikeda, R. Slow Mode-Based Control Method for Multi-Point Temperature Control System. *Processes* **2019**, *7*, 533. [CrossRef]
8. Hu, B.; Yang, J.; Li, J.; Li, S.; Bai, H. Intelligent Control Strategy for Transient Response of a Variable Geometry Turbocharger System Based on Deep Reinforcement Learning. *Processes* **2019**, *7*, 601. [CrossRef]
9. Pulido, B.; Zamarreño, J.M.; Merino, A.; Bregon, A. State space neural networks and model-decomposition methods for fault diagnosis of complex industrial systems. *Eng. Appl. Artif. Intell.* **2019**, *79*, 67–86. [CrossRef]
10. Zhao, C.; Sun, H.; Tian, F. Total Variable Decomposition Based on Sparse Cointegration Analysis for Distributed Monitoring of Nonstationary Industrial Processes. *IEEE Trans. Control Syst. Technol.* **2019**, 1–8. [CrossRef]
11. Han, T.; Liu, C.; Yang, W.; Jiang, D. Deep transfer network with joint distribution adaptation: A new intelligent fault diagnosis framework for industry application. *ISA Trans.* **2019**, in press. [CrossRef] [PubMed]
12. Gao, Z.; Cecati, C.; Ding, S.X. A Survey of Fault Diagnosis and Fault-Tolerant Techniques Part II: Fault Diagnosis with Knowledge-Based and Hybrid/Active Approaches. *IEEE Trans. Ind. Electron.* **2015**, *62*, 1. [CrossRef]
13. Ji, K.; Lu, Y.; Liao, L.; Song, Z.; Wei, D. Prognostics Enabled Resilient Control for Model-based Building Automation Systems. In Proceedings of the 12th Conference of International Building Performance Simulation Association, Sydney, Australia, 14–16 November 2011.
14. Rieger, C.G.; Gertman, D.I.; McQueen, M.A. Resilient Control Systems: Next Generation Design Research. In Proceedings of the 2nd IEEE Conference on Human System Interaction, Catania, Italy, 21–23 May 2009; Volume 9, pp. 632–636.
15. Ibrahim, M. A resiliency measure for communication networks. In Proceedings of the 8th International Conference on Information Technology (ICIT), Amman, Jordan, 17–18 May 2017; pp. 151–156.
16. SEI. *Architecture Analysis and Design Language*; SEI: Pittsburgh, PA, USA, 2004; Available online: http://standards.sae.org/as5506/ (accessed on 29 October 2019).
17. Rockwell-Collins; University of Minnesota. The Assume Guarantee Reasoning Environment. 2016. Available online: http://loonwerks.com/tools/agree.html (accessed on 29 October 2019).
18. Sheeran, M.; Singh, S.; Stålmarck, G. Checking Safety Properties Using Induction and a SAT-Solver. In *Proceedings of the Computer Vision—ECCV 2012; Austin, TX, USA, 1–3 November 2000*; Springer Science and Business Media LLC: Berlin, Germany, 2000; Volume 1954, pp. 127–144.
19. Download Unity, Unity3d. Available online: https://unity3d.com/get-unity/download (accessed on 29 October 2019).
20. Ibrahim, M.; Alsheikh, A. Assessing Level of Resilience Using Attack Graphs. In Proceedings of the 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Iasi, Romania, 28–30 June 2018; pp. 1–6.
21. Sheyner, O.; Wing, J. Tools for generating and analyzing attack graphs. In *Proceedings of the International Symposium on Formal Methods for Components and Objects*; Leiden, Germany, 4–7 November 2003, Springer: Berlin, Germany, 2003.
22. Ou, X.; Anoop, S. Attack graph techniques. In *Quantitative Security Risk Assessment of Enterprise Networks*; Springer: New York, NY, USA, 2012; pp. 5–8.
23. Jajodia, S.; Noel, S.; Kalapa, P.; Albanese, M.; Williams, J. Cauldron mission-centric cyber situational awareness with defense in depth. In Proceedings of the 2011—MILCOM 2011 Military Communications Conference, Baltimore, MD, USA, 7–10 November 2011.
24. Martın, B.; Lupu, E.C. Naggen: A Network Attack Graph Generation Tool. In Proceedings of the IEEE Conference on Communications and Network Security, Las Vegas, NV, USA, 9–11 October 2017.

25. Ou, X.; Boyer, W.F.; McQueen, M.A. A scalable approach to attack graph generation. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; p. 336.
26. Somesh, J.; Sheyner, O.; Wing, J. Two formal analyses of attack graphs. In Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW-15), Cape Breton, NS, Canada, 24–26 June 2002.
27. Al Ghazo, A.T.; Ibrahim, M.; Ren, H.; Kumar, R. A2G2V: Automatic Attack Graph Generation and Visualization and Its Applications to Computer and SCADA Networks. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, 1–11. [CrossRef]
28. Wang, S.; Zhang, Z.; Kadobayashi, Y. Exploring attack graph for cost-benefit security hardening: A probabilistic approach. *Comput. Secur.* **2013**, *32*, 158–169. [CrossRef]
29. Huan, W. A Vulnerability Assessment Method in Industrial Internet of Things Based on Attack Graph and Maximum Flow. *IEEE Access* **2018**, *6*, 8599–8609.
30. Shandilya, V.; Simmons, C.B.; Shiva, S. Use of Attack Graphs in Security Systems. *J. Comput. Netw. Commun.* **2014**, *2014*, 1–13. [CrossRef]
31. Lippmann, R.P.; Ingols, K.W. *An Annotated Review of Past Papers on Attack Graphs*; Project Report IA-1; Massachusetts Institute of Technology, Lincoln Laboratory: Lexington, MA, USA, 2005.
32. Louthan, G.; Michael, H.; Phoebe, H.; Peter, H.; John, H. Hybrid extensions for stateful attack graphs. In Proceedings of the 9th Annual Cyber and Information Security Research Conference, Oak Ridge, TN, USA, 8–10 April 2014; p. 101.
33. Louthan, G.; Phoebe, H.; Peter, H.; John, H. Toward hybrid attack dependency graphs. In Proceedings of the 7th Annual Workshop on Cyber Security and Information Intelligence Research, Oak Ridge, TN, USA, 12–14 October 2011.
34. Hawrylak, P.J.; Haney, M.; Papa, M.; Hale, J. Using hybrid attack graphs to model cyber-physical attacks in the Smart Grid. In Proceedings of the 5th International Symposium on Resilient Control Systems (ISRCS), Salt Lake City, UT, USA, 14–16 August 2012; pp. 161–164.
35. Nichols, W.; Hawrylak, P.; Hale, J.; Papa, M. Introducing priority into hybrid attack graphs. In Proceedings of the 12th Annual Conference on Cyber and Information Security Research, Oak Ridge, TN, USA, 4–6 April 2017.
36. Chen, Y.-C.; Gieseking, T.; Campbell, D.; Mooney, V.; Grijalva, S. A Hybrid Attack Model for Cyber-Physical Security Assessment in Electricity Grid. In Proceedings of the 2019 IEEE Texas Power and Energy Conference (TPEC), College Station, TX, USA, 7–8 February 2019; pp. 1–6.
37. C. N. Academy. Routing Protocols and Concepts. Available online: https://www.netacad.com/web/aboutus/ccna-exploration (accessed on 29 October 2019).
38. An Infinite-State Model Checker for Safety Properties. Loonwerks. Available online: http://loonwerks.com/tools/jkind.html (accessed on 29 October 2019).
39. Halbwachs, N.; Paul, C.; Pascal, R.; Daniel, P. The synchronous data flow programming language LUSTRE. *Proc. IEEE* **1991**, *79*, 1305–1320. [CrossRef]
40. Carnegie-Mellon-University. Open Source AADL Tool Environment for the SAE Architecture. 2018. Available online: http://osate.github.io/index.html (accessed on 29 October 2019).
41. Craighead, J.; Burke, J. Using the unity game engine to develop SARGE: A case study. In Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008), Nice, France, 22–26 September 2008.
42. Àlvarez, C.; Blesa, M.J.; Serna, M. The robustness of stability under link and node failures. *Theor. Comput. Sci.* **2011**, *412*, 6855–6878. [CrossRef]
43. Riverbed Technology Inc. Opnet Modeler. Available online: http://mediacms.riverbed.com/documents/download.html (accessed on 29 October 2019).
44. HTML5 Web Development Support. NetBeans. Available online: https://netbeans.org/features/html5/index.html (accessed on 29 October 2019).
45. Böck, H. *The Definitive Guide to NetBeans™ Platform 7*; Apress: New York, NY, USA, 2011.