A2G2V: Automatic Attack Graph Generation and Visualization and Its Applications to Computer and SCADA Networks*

Alaa Al Ghazo[1,2], Ratnesh Kumar[2]

[1]Dept. Electrical and Computer Engineering
University of Hartford, West Hartford, CT 06117

[2] Dept. Electrical And Computer Engineering
Iowa State University, Ames, IA 50010

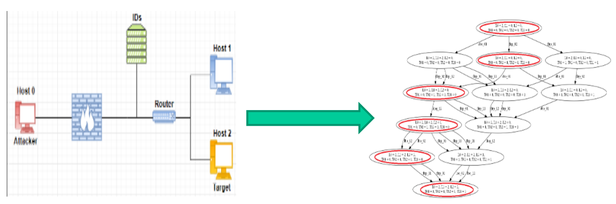
Abstract

Securing Cyber-Physical systems (CPS), and Internet of things (IoT) systems requires the identification of how interdependence among existing atomic vulnerabilities may be exploited by an adversary to stitch together an attack that can compromise the system. Therefore, accurate attack-graphs play a significant role in systems security. A manual construction of the attack-graphs is tedious and error-prone, this paper proposes a model-checking based Automated Attack-Graph Generator and Visualizer (A2G2V). The proposed A2G2V algorithm uses existing model-checking tools, an architecture description tool, and our own 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 architecture description tool captures a formal representation of the networked system, its atomic vulnerabilities, their pre- and post- conditions, and security property of interest. A model-checker is employed to automatically identify an attack sequence in form of a counterexample. Our own code integrated with the model-checker parses the counterexamples, encodes those for specification relaxation, and iterates until all attack sequences are revealed. Finally, a visualization tool has also been incorporated with A2G2V to generate a graphical representation of the generated attack-graph. The results are illustrated through application to computer as well as control (SCADA) networks

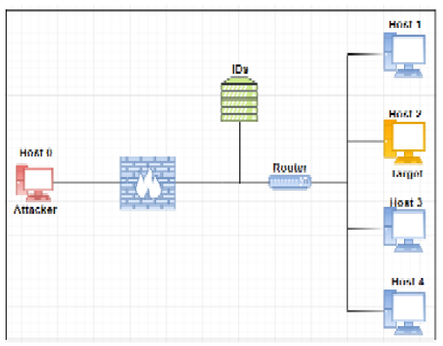
Background

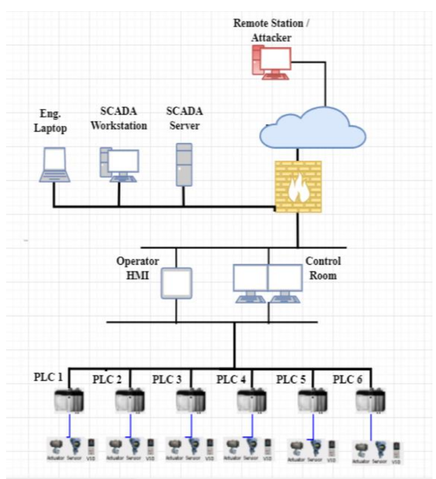
- ICS/SCADA Control and supervise all critical infrastructure, such as power grid, nuclear plants, oil and gas refinement, and water distribution
- ICS/SCADA architectures use devices (PLCs/RTUs), network protocols (MODBUS/ PROFIBUS), and graphical user interfaces for high-level supervisory control
- Cloud computing and IoT revolutions have led ICS/SCADA connections to the internet/cloud
- Internet connection introduced cybersecurity vulnerabilities and threats to ICS/SCADA

Attack-Graph

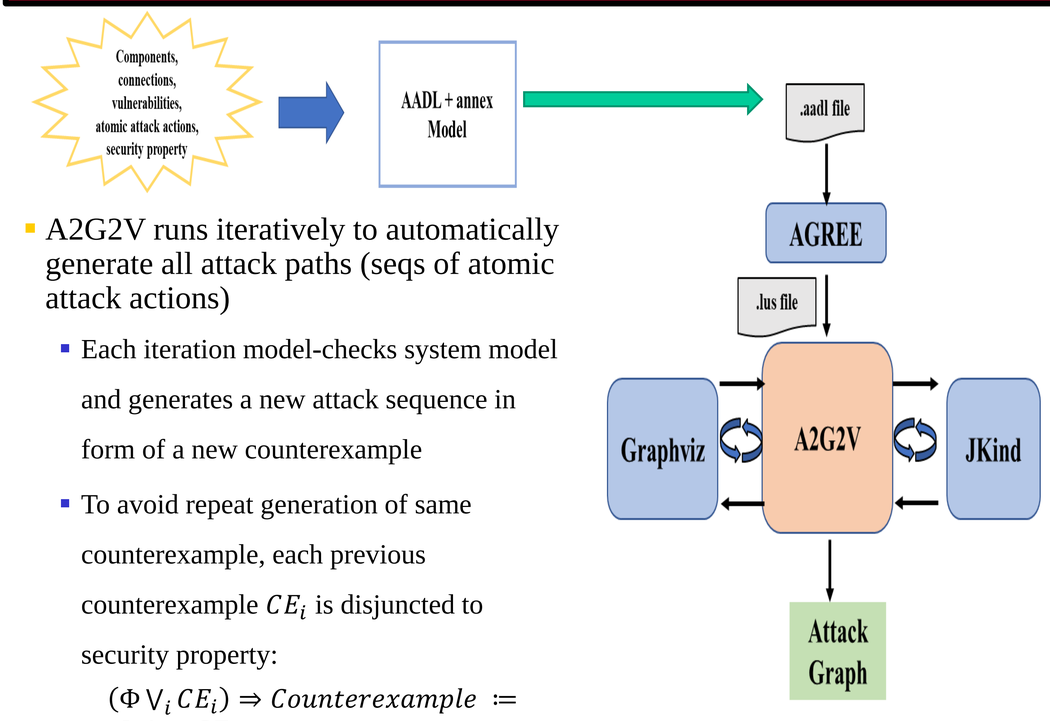


- Requires comprehensive overview of
  - System architecture --- components and connectivity
  - Assets/Services and Protections
  - Vulnerabilities and Threats/Attacks
- Constructed by a state space representation and exploration:
  - Identify dynamic variables and their evolution under atomic attacks
  - Explore state-space to list executions that violate security property of interest
  - Attack graph is union of all such paths





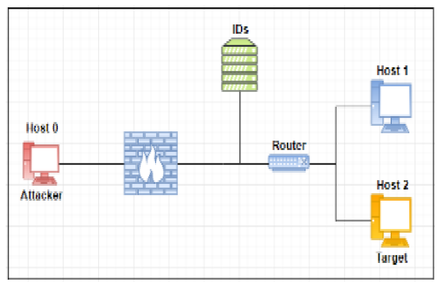
A2G2V

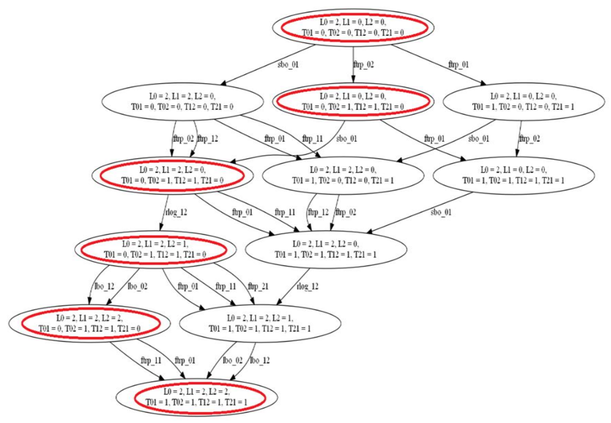


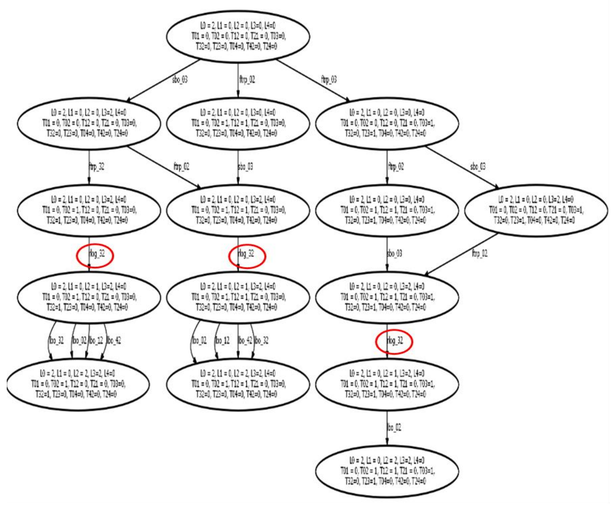
- A2G2V runs iteratively to automatically generate all attack paths (seqs of atomic attack actions)
  - Each iteration model-checks system model and generates a new attack sequence in form of a new counterexample
  - To avoid repeat generation of same counterexample, each previous counterexample CE_i is disjuncted to security property:

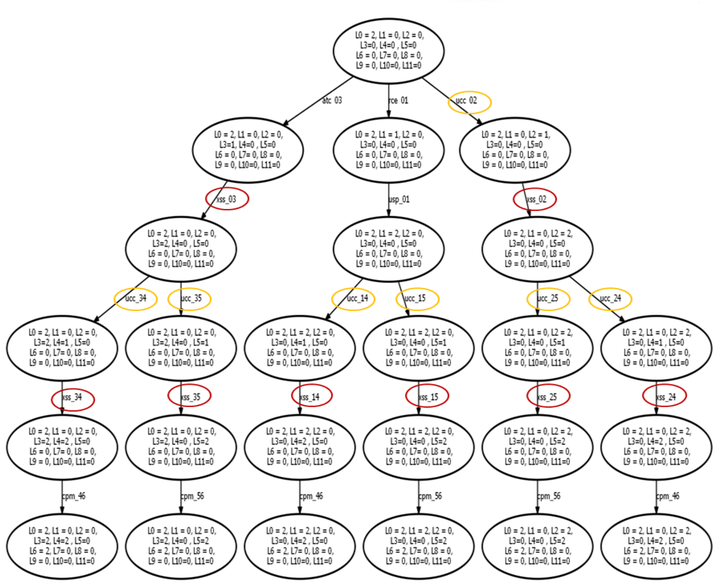
    $(\Phi \bigvee_i CE_i) \Rightarrow Counterexample \coloneqq \neg\Phi \bigwedge_i \neg CE_i$
  - The algorithm terminates once all acyclic attack paths have been generated.

Case study

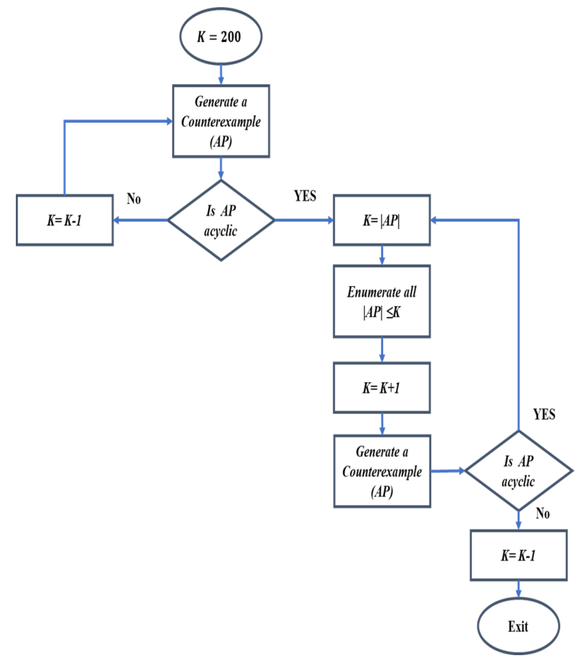








A2G2V Algorithm



- The goal is to find all acyclic attack paths that an attacker can use to compromise the system
- Acyclic counterexamples length is upper-bounded by the "depth" of the system model
- Each iteration Jkind uses BMC to limit the generated counterexample length (K)

Summary

We presented a first general model-based automated attack graph generator and visualizer algorithm and its C-based implementation tool A2G2V, and also illustrated it through three examples. The key to automation is the employment of an architectural description language to capture the security-related details of a networked system, an automated encoding of the latest counterexample to relax the current specification, and an iterative adjustment of the search depth of a bounded model-checker to identify all the acyclic counterexamples. Our algorithm formally models the system using AADL, iteratively model-checks the system with JKind model-checker to generate attack paths, and combines the attack paths into an attack-graph using GraphViz

* This paper was published in IEEE SMC TRANSACTIONS