Hybrid Extensions for Stateful Attack Graphs

George Louthan
Tandy Supercomputing Center
Tulsa, OK, USA
george@tandysupercomputing.org

Michael Haney
The University of Tulsa
Tulsa, OK, USA
michael-haney@utulsa.edu

Phoebe Hardwicke
The University of Tulsa
Tulsa, OK, USA
Phoebe-hardwicke@utulsa.edu

Peter Hawrylak
The University of Tulsa
Tulsa, OK, USA
peter-hawrylak@utulsa.edu

John Hale
The University of Tulsa
Tulsa, OK, USA
john-hale@utulsa.edu

ABSTRACT

Critical infrastructures and safety critical systems increasingly rely on the carefully orchestrated interactions between computers, networks and kinetic elements. The dominant formalisms for modeling such hybrid systems (those with discrete and continuous components) are geared towards simple reactive systems working in isolation. By contrast, modern cyber-physical systems depend on highly interconnected computational components and often function in potentially hostile environments. This paper describes linguistic and type extensions to the stateful attack graph, which models the functional nature of attacks on purely discrete information systems, to include continuous system elements and time evolution. The resulting formalism is called the hybrid attack graph, which captures an integrated view of the vulnerability space between information systems and a restricted but useful set of hybrid systems.

1. INTRODUCTION

Hybrid systems bridge the divide between the computational and the physical, blending discrete and continuous elements into a single formal framework. When linked with a significant network component, these systems are referred to as cyber-physical systems (CPSs), which abound in safety-critical domains such as medical, manufacturing, automotive, and critical infrastructure.

This paper explores one avenue for modeling CPSs for security analysis. Traditional formal models of purely discrete computer networks are inappropriate for these systems because of their inability to capture the continuous domain; they also typically lack a robust notion of time. On the other hand, modeling methods from the world of isolated control systems cannot model the complex distributed networks that are the hallmark of CPSs.

This paper presents linguistic and type extensions for hybrid attack graphs (HAGs), a formalism that excels at capturing complex interdependencies of a hybrid system's vulnerability surface. HAGs incorporate the continuous domain, the goal being an integrated view of a CPS's attack space.

The remainder of this paper is as follows. Section 2 provides background in hybrid systems modeling and attack graphs. Section 3 introduces the discrete attack graph model extended by this paper and presents an example where a cyber-attack against an automobile is modeled. Section 4 provides conclusions and recommends future work.

CISR '14, Apr 08-10 2014, Oak Ridge, TN, USA
ACM 978-1-4503-2812-8/14/04.
http://dx.doi.org/10.1145/2602087.2602106

2. BACKGROUND

2.1 Hybrid Systems

The term hybrid system often refers to programmed control systems that interact with the physical world [1]. A CPS is a highly networked hybrid system.

A valuable formalism for modeling hybrid systems is the hybrid automaton [1], which pairs a finite state machine with differential equations that govern its continuous-domain behavior. Formally, a hybrid automaton, H, is made up of a set, X, of real-valued state variables, a set of their first derivatives, a set of operational modes and guarded switches between modes, and predicates describing continuous evolution over time. One can think of a hybrid automaton as a pairing of a finite state machine whose states ("modes") and transitions ("switches") denote the discrete-domain behavior of the system with differential equations governing continuous-domain behavior.

Modes may be decorated with invariant conditions (when the system may be in that mode), flow conditions (how the continuous state variables may evolve while in that mode), and initial conditions (when and if the automaton may start in that mode). Switches are decorated with jump conditions, which determine (1) when the switch is allowed to be taken, and (2) the discrete changes in state variables due to that switch's activation.

A hybrid automaton is not guaranteed to have a valid execution, and computing whether it does or not is nontrivial [2]. Model checking has been developed for some subclasses of automata [3, 4], and many useful properties of them are undecidable [5].

Moreover, hybrid automata provide only a rudimentary notion of communication unsuitable for modeling real computer networks. Other hybrid systems frameworks such as hybrid process algebras [6, 7] and hybrid Petri nets [8] share similar drawbacks. Attempts have been made to improve the hybrid automaton's network modeling capability. The designation of shared actions and shared variables as "input" or "output" is a popular tactic, as in the hybrid I/O automaton [9, 10].

2.2 Attack Graphs

Attack graphs engage a graph theoretic model to permit a network topology aware exploration of the state space of a system. In an attack graph, vertices represent individual system states, and edges represent state transitions caused by an adversary's activities. The concept as originally introduced shares with all modern incarnations notions of attack patterns (complete with free varia-

bles) to be bound to state transitions; network elements and their individual configurations; and network topology [11].

Most approaches to attack graph modeling represent exploits using preconditions and postconditions [12, 13]. The approaches to modeling of stateful attack graphs can be distinguished by the philosophy that guides the representation of the underlying network model. Specification of the underlying network model may be done with very loose restrictions, allowing arbitrary keywords as named qualities and topologies of network objects, as is favored in [14, 15]. We employ this method, which permits a more straightforward adaptation into the continuous domain.

The alternative is domain-driven, confining the modeler to terms that impose explicit computer networking concepts onto the model [13]. This allows generation and analysis to take a more nuanced view of a network state, supporting reachability analysis to determine whether a given topology admits communication between two hosts [16].

Attack graph generation is the process of chaining exploits to enumerate the attack space [11, 17, 18, 19]. Modern methods for generating attack graphs share a common architecture: the attack graph generation process accepts network state and exploit patterns as input, applying exploit postconditions back onto the network state to generate its output of successor states. These models can be used for evaluating network security [14, 20], offering mitigation recommendations [15], and intrusion detection system integration [21]. Hybrid variants of attack graphs have been developed [22, 23, 24]. This paper describes the linguistic and type extensions required to characterize system models for HAG generation.

3. HYBRID ATTACK GRAPHS (HAGs)

3.1 Modeling Framework

The HAGs presented in this paper consist of states (or network model instances), encompassing an asset list and a collection of facts, and an exploit framework made up of function-like free exploits and bound attacks. The network model for the initial state is expressed as an input in a specification language.

The first component of the specification is an asset list. Next, the initial fact base is specified as a semicolon separated list of facts. Quality facts are simple string key/value pairs assigned to a single asset each. Topologies are specified as unidirectional ("->") or bidirectional ("<->"), although the bidirectional topology is actually compiled into two distinct unidirectional topology facts. Platform facts adopt the format used by MITRE's Common Platform Enumeration (CPE) [25], reproduced below:

cpe:/part:[vendor]:[product]:[version]:

[update]:[edition]:[language]

Exploit patterns resemble functions whose bodies are lists of preconditions, which are identical in form to the facts in the network model. Preconditions document those properties that must be in place for an exploit to bind (execute) to a given state. Postconditions are simply deletions or insertions (with overwrite semantics) of facts that result in a change of state.

For a variety of modeling tasks, it is convenient to cause an exploit to be fired on all possible bindings simultaneously or to synchronize multiple exploits. Two optional keywords are permitted at the beginning of the exploit header for this purpose: global and group(). The group keyword takes a single string argument: the group name.

When a global exploit is fired on one binding of assets, it will also be fired on all other assets for which a valid binding exists. When a group exploit is fired on a binding of assets, all other exploits (which must have the same arity) in that group will also attempt the same asset binding. In both cases, the result is a single edge on the attack graph.

3.2 Hybrid Extensions

The objective of the HAG is to capture blended attack vectors (comprising discrete and continuous exploit events) in CPSs and hybrid systems. As in hybrid automata, a hallmark of hybrid systems is that their discrete behavior places them into a mode in which the passage of time causes some kind of state evolution. Likewise, a hybrid or blended attack takes the form of a discrete action within a mode causing the system to evolve in some malicious way.

The hybrid extensions for stateful attack graphs presented in this paper address these considerations in two ways. The first is the purely mechanical addition of real valued continuous qualities and topologies. This includes a type system to differentiate them from discrete ("token valued") facts, while permitting real topologies only to take values. The second contribution is a method for modeling the progression of time.

Types are assigned and enforced by operators. The sets of both assignment and relational operators disjoint between real and token facts. While token values are assigned with the = operator, real facts are assigned using a := operator. Token facts are tested using = and !=. The relational operators permitted for real facts are ==, >, >=, <=, and <> (for not equal). New C-like assignment operators are allowed for real facts: :=, +=, -=, *=, and /=.

A fundamental requirement of modeling hybrid systems is capturing the progression of time. To that end, we introduce a practical method for handling time using only existing attack graph functionality. Here, 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 (analogous to a hybrid automaton's mode). Their "globalness" triggers on all assets simultaneously, while the grouping causes all the time exploits to trigger in concert. This effectively discretizes the time domain.

A selection of time exploits is provided in Figure 1 that, when paired with the network specification in Figure 2, models a car driving away from a wall, unless compromised by an attacker who causes the car to drive toward the wall at a constant rate. The resulting HAG is shown in Figure 3. As the car is compromised from the start, it only has three states. Note the exploit transitions, both of which are collected into a group called "time," signifying that they represent time steps.

Suppose the initial conditions are changed so that the "compromised" quality begins as false and the own_civic exploit is employed to allow the attacker to take control of the car. In this case, the resulting HAG (shown in Figure 4 to a generation depth of 5) illustrates the interaction of the discrete behavior of the attacker (the "cyber" aspect) with the continuous evolution of the car (the physical aspect). This exemplifies the "mode" paradigm of time evolution in hybrid systems and attacks. In this case, the evolution of time is denoted by the transitions labeled group(time) in the exploits in Figure 1.

The authors have developed a reference implementation which produces HAGs given the necessary inputs of asset, topology, and fact declarations in the grammar described above. This tool auto-

mates the task of creating HAGs by compiling the inputs specified and making the appropriate connections. It is a web-enabled interface that leverages GraphViz processing to generate the HAG images.

```
global group(time) exploit car_depart(c,w)=
    preconditions :
        platform : c, cpe:/h:honda;
        quality : c, compromised != true;
        platform : w, cpe:/h::wall;
        quality : c, status=up;
    postconditions :
        update topology : c<->w, distance+=25;
.

global group(time) exploit car_approach(c,w)=
    preconditions :
        platform : c, cpe:/h:honda;
        quality : c, compromised=true;
        platform : w, cpe:/h::wall;
        quality : c, status=up;
        topology : c<->w, distance>25;
    postconditions :
        update topology : c<->w, distance-=25;
.

global group(time) exploit car_crash(c,w)=
    preconditions :
        platform : c, cpe:/h:honda;
        quality : c, compromised=true;
        platform : w, cpe:/h::wall;
        quality : c, status=up;
        topology : c<->w, distance<=25;
    postconditions :
        update topology : c<->w, distance:=0;
        update quality : c, status=down;
.

exploit own_civic(c)=
    preconditions:
        platform : c, cpe:/h:honda:civic;
        quality : c, compromised=false;
        quality : c, status=up;
    postconditions:
        update quality : c, compromised=true;
.
```

Figure 1: Car example exploit patterns.

```
network model=
    assets :
        civic;
        wall;
    facts:
        platform : civic, cpe:/h:honda:civic;
        quality : civic, compromised=true;
        quality : civic, status=up;

        platform : wall,cpe:/h::wall;

        topology : civic<->wall, distance:=50;
.
```

Figure 2: Car example network model.

Besides the assistance provided by automating this process, the primary benefit of this prototype is intended to be the collaboration with domain experts who can provide the details necessary for an analyst to model hybrid and CPSs. This will allow the analyst to share the produced attack graph to discuss the nature of any systemic vulnerabilities which may be discovered. It allows the formal model and language to be more accessible to domain experts such as CPS engineers, greatly expanded the adoption of this powerful modeling technique.

4. CONCLUSIONS AND FUTURE WORK

This paper presents a set of linguistic and type extensions to attack graphs for modeling of hybrid systems and their evolution over time in the context of exploit interactions. This represents a new capability for attack graph modeling with respect to expressing a class of critical systems.

A significant challenge for the widespread use of attack graphs and HAGs is model acquisition. While a substantial investment of research effort has been made on pursuing the acquisition of information system models, there are new research opportunities on the physical side of CPSs model acquisition.

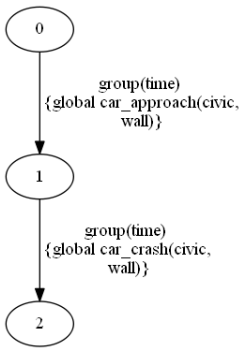

Figure 3: Simple car attack graph.

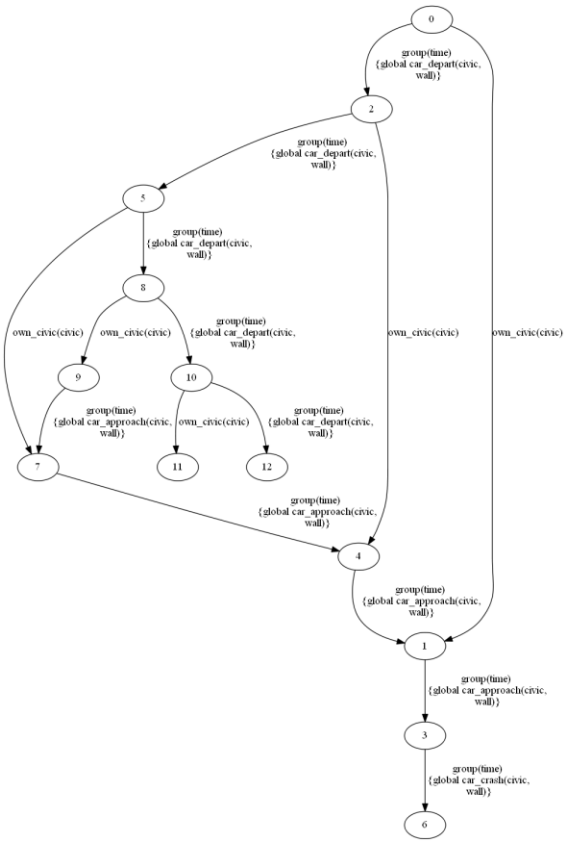

Figure 4: One-car automotive attack graph.

The worst-case execution time of the reference implementation of the HAG generator scales factorially with input assets and exponentially with maximum depth. This lags behind the state of the art, and using this tool, generation of attack graphs from nontrivial scenarios (with tens of hybrid assets) in acceptable time is daunting. However, the performance of the HAG generator is no worse than the discrete-case only implementation, which it extends. We believe the techniques that have been applied to the state of the art discrete-case attack graph generators to reduce their order can also be applied to this HAG generator to obtain similar results.

There are a variety of systematic exploit pattern processing enhancements that would prove useful. These include generic wildcard expressions in preconditions permitting hierarchies of network localities that more accurately reflect the real world; “pivot” exploits to transparently model attackers’ using a compromised

asset to attack its neighbors that are otherwise inaccessible to the attacker; and variables in exploit conditions.

Finally, one route for future work is the attack dependency graph [26, 27]. These are smaller, more efficiently generated, and easier to read than their stateful counterparts; furthermore, a hybrid version we propose in [22] exhibits some desirable properties with respect to modeling time evolution.

5. ACKNOWLEDGEMENT

This material is based on research sponsored by DARPA under agreement number FA8750-09-1-0208. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

6. REFERENCES

[1] Alur, R., C. Courcoubetis, T. Henzinger, and P. Ho. 1993. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In Hybrid Systems, Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel (Eds.). Springer-Verlag, London, UK, UK, 209-229.

[2] Lygeros, J., K.H. Johansson, S. Sastry, and M. Egerstedt. 1999. On the existence of executions of hybrid automata. In *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, 3:2249-2254. IEEE,.

[3] Frehse, G. 2005. PHAVer: Algorithmic verification of hybrid systems past HyTech. *Hybrid Systems: Computation and Control*, pg. 258-273.

[4] Henzinger, T.A., P.H. Ho, and H. Wong-Toi. 1997. HyTech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1, no. 1:110-122.

[5] Henzinger, T.A., P.W. Kopke, A. Puri, and P. Varaiya. 1998. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57, no. 1:94-124.

[6] Bergstra, J.A., and CA Middelburg. 2005. Process algebra for hybrid systems. *Theoretical Computer Science*, 335 no. 2-3:215-280.

[7] Cuijpers, P.J.L., and M.A. Reniers. 2005. Hybrid process algebra. *Journal of Logic and Algebraic Programming*, 62, no. 2:191-245.

[8] Champagnat, R., P. Esteban, H. Pingaud, and R. Valette. 1998. Petri net based modeling of hybrid systems. *Computers in industry*, 36, no. 1-2:139-146.

[9] Lynch, N., R. Segala, and F. Vaandrager. 2001. Hybrid I/O automata revisited. *Hybrid Systems: Computation and Control*, pg. 403-417.

[10] Lynch, N., R. Segala, F. Vaandrager, and H. Weinberg. 1996. Hybrid I/O automata. *Hybrid Systems III*, pg. 496-510.

[11] Phillips, C., and L.P. Swiler. 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pg. 71-79. ACM.

[12] Lippmann, R.P., K.W. Ingols, and MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB. 2005. *An annotated review of past papers on attack graphs*. Massachusetts Institute of Technology, Lincoln Laboratory.

[13] Templeton, S.J., and K. Levitt. 2001. A requires/provides model for computer attacks. In *Proceedings of the 2000 workshop on New security paradigms*, pg. 31-38, ACM.

[14] Ammann, P., D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217-224. ACM, 2002.

[15] Wang, L., S. Noel, and S. Jajodia. 2006. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29, no. 18:3812-3824.

[16] Ingols, K., M. Chu, R. Lippmann, S. Webster, and S. Boyer. 2009. Modeling modern network attacks and countermeasures using attack graphs. In *2009 Annual Computer Security Applications Conference*, pg. 117-126.

[17] Campbell, C., J. Dawkins, B. Pollet, K. Fitch, J. Hale, and M. Papa. 2002 On Modeling Computer Networks for Vulnerability Analysis. *DBSec*, 128:233-244.

[18] Hawrylak, P.J., G. Louthan, J. Daily, J. Hale, and M. Papa. 2012. Attack graphs and scenario driven wireless computer network defense. In *Situational Awareness in Computer Network Defense: Principles, Methods and Applications*, ed. C. Onwubiko, and T. Owens, Hershey, PA: IGI Global.

[19] Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J.M. Wing. 2002. Automated generation and analysis of attack graphs.

[20] Louthan, G., W. Roberts, M. Butler, and J. Hale. 2010. The blunderdome: an offensive exercise for building network, systems, and web security awareness. In *Proceedings of the 3rd international conference on Cyber security experimentation and test*, pg. 1-7. USENIX Association.

[21] Tidwell, T., R. Larson, K. Fitch, and J. Hale. 2001. Modeling internet attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and security*, 59.

[22] Louthan, G., P. Hardwicke, P. Hawrylak, and J. Hale. 2011. Toward hybrid attack dependency graphs. In *Proc. Cyber Security and Information Intelligence Research Workshop*.

[23] Hawrylak, P.J., Haney, M., Papa, M., and Hale, J., "Using hybrid attack graphs to model cyber-physical attacks in the Smart Grid," *Resilient Control Systems (ISRCS), 2012 5th International Symposium on*, pp. 161-164, 14-16 Aug. 2012.

[24] Hawrylak, P.J., Hartney, C., Papa, M., and Hale, J. 2013. Using Hybrid Attack Graphs to Model and Analyze Attacks against the Critical Information Infrastructure. In Critical Information Infrastructure Protection and Resilience in the ICT Sector, ed. P. Théron and S. Bologna, Hershey, PA: IGI Global.

[25] Buttner, A. and N. Ziring. 2009. Common platform enumeration (cpe) | specification, March.

[26] Jajodia, S., S. Noel, and B. O'Berry. 2005. Topological analysis of network attack vulnerability. *Managing Cyber Threats*, pg. 247-266.

[27] Noel, S., and S. Jajodia. 2004. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pg. 109-118.