Toward Hybrid Attack Dependency Graphs

[Extended Abstract]

George Louthan
The University of Tulsa
800 S. Tucker Drive
Tulsa, Oklahoma 74104
george-louthan@utulsa.edu

Phoebe Hardwicke
The University of Tulsa
800 S. Tucker Drive
Tulsa, Oklahoma 74104
phoebe-hardwicke@utulsa.edu

Peter Hawrylak
The University of Tulsa
800 S. Tucker Drive
Tulsa, Oklahoma 74104
peter-hawrylak@utulsa.edu

John Hale
The University of Tulsa
800 S. Tucker Drive
Tulsa, Oklahoma 74104
john-hale@utulsa.edu

ABSTRACT

This extended abstract presents a set of continuous-domain extensions to the attack graph, a formalism used to model the interactions of multiple exploits and assets in a network. These extensions result in a new modeling framework called the *hybrid attack dependency graph*, which provides the novel capability of modeling continuous state variables and their evolution over the execution of attacks with duration.

1. INTRODUCTION

As computers' interactions with each other and the physical world become pervasive, they are being targeted for exploitation in new, dangerous, and sometimes dramatic ways. Systems implementing such interactions, termed *hybrid systems* or in some highly networked cases *cyber-physical systems*, are widespread in safety-critical medical, infrastructure, automotive, and other domains.

This publication is concerned with the specification and generation of theoretical models for cyber-physical systems security. Specifically, we present the *hybrid attack dependency graph*, an early stage extension of a discrete domain formalism called the attack graph that permits modeling of continuous valued properties, such as those often in play in the physical world.

According to the 2008 Report of the Cyber-Physical Systems Summit, "The principal barrier to developing CPS is the lack of a theory that comprehends cyber and physical resources in a single unified framework." [10] The summit further identified as part of the necessary scientific and technological foundations of cyber physical systems both (1) new modeling frameworks and (2) "theories of cyber-physical inter-dependence" [10]. This work contributes to both of those research needs.

This paper is concerned with proposing structure and preliminary applications of the hybrid attack dependency graph. Although there is value in discussing their generation from a network model, exploit set, and starting conditions, that is beyond the scope of this brief abstract.

The remainder of this paper is organized as follows. Section 2 provides background into cyber physical systems and attack graphs; section 3 introduces the proposed hybrid attack dependency graph and its nomenclature; section 4 gives an example of its use to model a particular attack and expands on its properties; and section 5 draws conclusions and proposes future work.

2. BACKGROUND

2.1 Hybrid and Cyber-physical Systems

A system with both continuous (frequently physical) components and discrete (frequently digital) components is said to be a *hybrid system*, named for its characteristic blending of the two domains. Examples of hybrid computer systems abound in industrial controls and critical infrastructure, for example.

The term hybrid system is an older one that was coined as researchers began to model the newly pervasive reactive systems that arose as programmed control of the physical world became widespread [1]. However, today's most critical and exposed systems have an additional feature: extensive networking. For this reason, we say that we are concerned with modeling security of *cyber-physical systems* (CPS), networked computer systems that are tightly coupled to the physical world.

2.2 Attack Graphs

The attack graph is one of several formalisms that use graph theory to model attacks and their interactions on networks of computer systems. As introduced originally in 1998, the attack graph models a system's security-relevant state space, in which vertices represent system states and edges represent state transitions, typically caused by the actions of

CSIIRW '11, October 12-14, Oak Ridge, Tennessee, USA

an adversary. Their use permits an exploration of a system's state space in a manner that takes into account the interactions among vulnerabilities on interconnected systems [12].

From the beginning, the framework has used several fundamental concepts in the generation of attack graphs: network elements and their configurations (e.g. platform or services), attack patterns (which could be bound to sets of network elements to produce state transitions), and network topology.

Modern incarnations of attack graphs tend to specify attack patterns as sets of preconditions and postconditions [9, 15]. Matching preconditions with the system model for a state node and mapping the postconditions back onto the model permits the identification of an attack's successor states. Attack graph generation is the process of chaining exploits to enumerate the attack space [5, 12, 14].

More recently, a variation of the attacks graph is being adopted by some researchers: building graphs of the dependencies of attacks upon each other called *exploit dependency graphs* or *attack dependency graphs* (our preferred nomenclature) [7, 11]. These smaller, more efficiently generated models encode equivalent information and eliminate redundancy by trading one common graph application for another: they replace the state transition graph with a dependency graph.

In a traditional attack graph, the nodes are states, and the edges are transitions, meaning each edge is a bound attack. In an attack dependency graph, edges represent one-way dependencies between *two* types of nodes: conditions (representing properties of the network state as attack preconditions or postconditions) and attacks (representing the bound attacks themselves). For each edge, the destination node is said to depend upon its source node. Also permitted are "and" and "or" semantics.

The most important concepts in our attack graph model are as follows. *Assets* are the security principals in the system, representing hosts, people, and other "nouns." *Facts* are statements about the properties of assets: either *platform* facts, which encode a Common Platform Enumeration (See [4]) description of an asset, named *quality* facts, which store an arbitrary value about a single asset, and named *topology* facts, which describe unidirectional relationships between two assets (topologies may only take real values; otherwise they are only named). *Exploits* encode preconditions and postconditions in the form of these facts. When compiled into an attack dependency graph, possible facts about assets become condition nodes, and exploits bound to assets become attack nodes.

In the attack dependency graphs presented here, attacks are represented as boxed nodes, initial conditions are surrounded by pentagons, and other reachable conditions are bare text. Multiple edges incident upon one node are said to represent an "or" condition unless grouped by an "and" node.

A very simple sample attack dependency graph is provided in Fig. 1. The scenario involves three assets: an attacker, a web server, and a printer. The attacker has remote web access to the server, and the server is connected to the printer. An attack called `root_server` models some server vulnerability allowing the attacker to gain administrative access to the server. Two other attacks are possible from here: denial of service to the web server, and instructing the printer to change its ready message to "Out of toner."

For reasons addressed in the next section, this type of attack graph is a natural fit for our hybrid enhancements.

3. HYBRID ADGS

A characteristic of a hybrid attack is that it takes a range of real values as preconditions and output a range of values as postconditions, as in an attack that drains energy, increases the speed of a vehicle or centrifuge, and acts to change other values over time. As articulated by Lee, "In the physical world, the passage of time is inexorable and concurrency is intrinsic" [8]. The passage of time is difficult to model in all discrete attack graphs, and concurrency is all but impossible to model in a traditional state transition attack graph.

These problems are solved with only two structural additions, described in section 3.1. These permit a surprising array of new features and capabilities.

3.1 Additions

The first addition is a new type of condition node, which represents a continuous range of possible values that a real valued fact (quality or topology) may hold. These are labeled with the standard interval notation and may be open, closed, or half-open/half-closed (e.g. quality $\in (0, 100]$) and may also be unbounded (e.g. distance $\in [100, +\infty)$). According to the initial conditions of the system and the exploits in play, the generation process must automatically divide the full range of possible values for each quality or topology into the disjoint ranges necessary to express dependencies and reachable conditions. For example, if an energy level can vary from 0 to 100 inclusive but the system's exploit dynamics depend only upon whether energy remains, and its initial energy level is 79, then its full range of $[0, 100]$ may be split into four condition nodes: $t = 0$ and $t \in (0, 79)$, $t = 79$, and $t \in (79, 100]$.

The second addition is a new type of attack node that supports ranges of values in conditions as both preconditions and postconditions. This is actually implemented using a special kind of dependency edge, a double arrow decorated with t, representing the passage of time. The application of these types of attacks is not so much a discrete event as it is the placement of the system in an operational mode that, combined with the passage of time, would cause an evolution in the state of the system.

This addresses the shortcomings described in the previous section: attacks with ranges of real-valued conditions and durational application brings time into play, if perhaps not necessarily inexorably; and the elimination of sequential state transitions permits concurrency.

4. EXAMPLE

4.1 Introduction

This section models an example denial of service attack on the ISO 18000-7 RFID tag inventory system similar to those used by the United States Department of Defense for shipping tracking and the Department of Energy for tracking spent fuel containers [6]. The attack is similar to the ones described by Buennemeyer, *et al.* [3], and is of a newly distinguished class of attacks sometimes termed *denial of sleep attacks* [2, 13]. An energy draining attack to deplete the tags' batteries could significantly speed their loss of power and cause unnecessary radiation exposure for maintenance workers.

The ISO 18000-7 tags have two modes: an active mode, and a sleep mode in which their power consumption is significantly reduced. A denial of sleep attack occurs when the tag is not

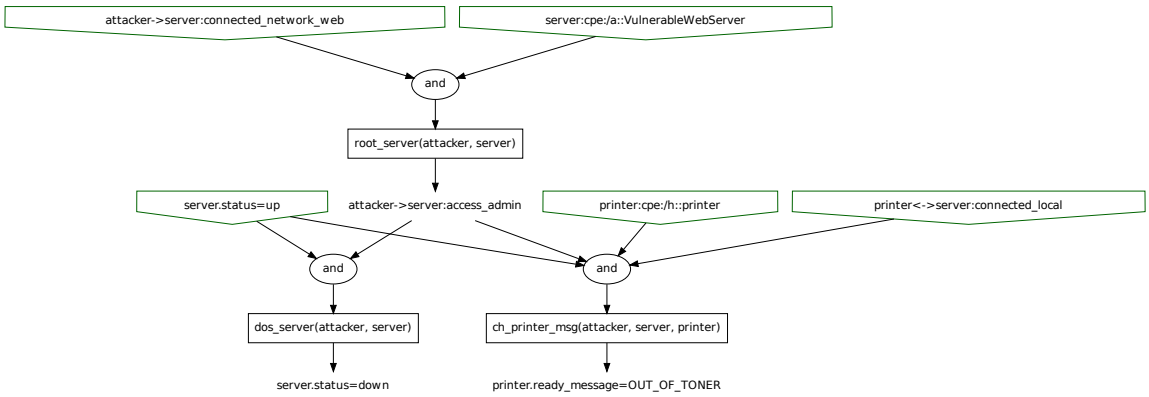

Figure 1: Discrete attack dependency graph

permitted to enter sleep mode or is awakened more frequently than normal, for example by a rogue reader or a compromised legitimate reader.

Let us consider two simple examples involving a single reader and a single tag. In the first, the attacker compromises the reader using a discrete attack called `own`, then accomplishes the denial of sleep attack.

The second, a somewhat more contrived case, adds a discrete attack to drain the last of the tag's power in a single action when it drops below power level 10. This could conceivably be accomplished by requesting a malformed or excessively expensive database query, for example, but the goal is purely to illustrate a more intimate mixing of discrete and continuous actions.

In both cases, the model uses power level 100 to represent full power and 0 to represent an empty battery.

4.2 HADG and Discussion

4.2.1 Hybrid Attacks

In the pure denial of sleep attack, the possible power levels are broken into three conditions: the starting power level, power levels above the starting level, power levels below the starting level (including 0).

This is modeled in Fig. 2, with the power level starting at 80. Note that the interval $(80, 100]$ is not included in this graph, as even though it is a valid interval, it is unreachable in the given scenario.

4.2.2 Mixed Hybrid and Discrete Attacks

The second denial of sleep example in Fig. 3 (which excludes the `own` step for space reasons) provides a more nuanced view of the characteristics of the hybrid attack dependency graph. Note that the $[0, 100]$ interval of possible battery values has been split into 80, the starting value; $(10, 80)$, the interval that can be achieved by executing the basic sleep denial attack and nothing else; $(0, 10]$, the interval that is likewise achieved from only the sleep denial but which also permits the discrete `kill` attack; and 0. $(80, 100]$ is unreachable and thus is not included in the graph.

There are a couple of important issues illustrated by this example. First, it should be noted that the structure of the graph depends heavily upon the initial conditions, as they

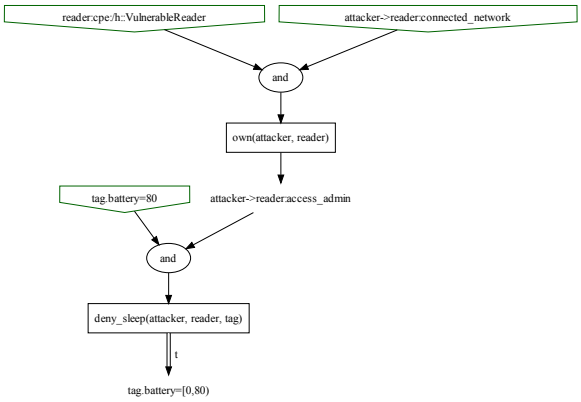

Figure 2: Pure denial of sleep attack

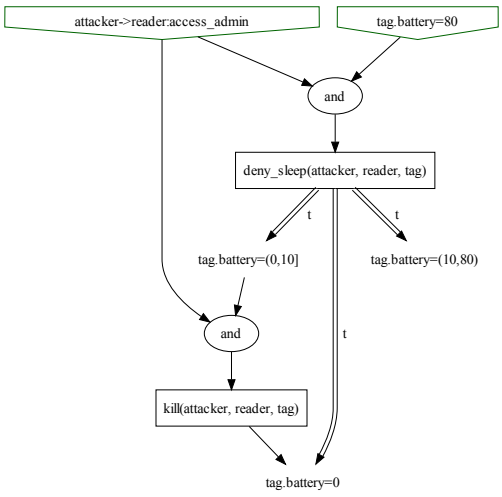

Figure 3: Hybrid denial of sleep attack

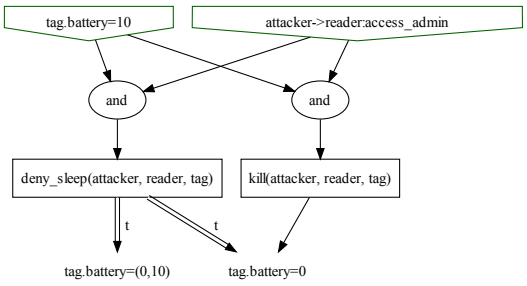

Figure 4: Hybrid denial of sleep attack (alternate starting conditions)

play a major role in the partitioning of the continuous state space into real intervals. For instance, if the starting value of `battery` had been 10, the graph would be as in Fig. 4.

Second, the question arises of why the node tag.battery $\in (0, 80)$ is a leaf node. The answer is simple, if not immediately obvious. The only possible action from the tag.battery $\in (0, 80)$ condition is `deny_sleep`, which has already been taken. The case where that attack is executed to bring the battery level to between 10 and 80, then executed again to bring the battery level lower is, without loss of generality, *already encoded* in the other postcondition nodes of the `deny_sleep` attack.

5. CONCLUSIONS AND FUTURE WORK

This abstract presents the hybrid attack dependency graph (HADG), which provides a set of continuous extensions to attack dependency graphs (exploit dependency graphs) with the goal of modeling cyber-physical systems.

A core characteristic of the HADG is the discretization into intervals of the reachable and relevant ranges of the system's state variables. These intervals are acted upon by both discrete attacks to cause discrete "jumps" and by continuous, durational attacks that place the system into a mode of operation that causes its continuous state to evolve over some time period. The reachable intervals, and their status as parents of other attacks, are heavily influenced by the starting state of the system.

In the short term, the process for generating HADGs must be articulated and formalized, and its performance characterized. A variety of analysis techniques that exist for exploit dependency graphs should be examined for suitability of adaptation to the hybrid space. A significant modeling effort is warranted to test the utility of the modeling framework itself.

6. ACKNOWLEDGMENT

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.

7. REFERENCES

[1] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid systems*, pages 209–229, 1993.

[2] M. Brownfield, Y. Gupta, and N. Davis. Wireless sensor network denial of sleep attack. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 356–364. IEEE, 2005.

[3] T. Buennemeyer, G. Jacoby, W. Chiang, R. Marchany, and J. Tront. Battery-sensing intrusion protection system. In *Information Assurance Workshop, 2006 IEEE*, pages 176–183. IEEE, 2006.

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

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

[6] K. Chen, H. Tsai, Y. Liu, and J. Shuler. A Radiofrequency Identification (RFID) Temperature-Monitoring System for Extended Maintenance of Nuclear Materials Packaging. In *Proceedings of 2009 ASME Pressure Vessels and Piping Division Conference, Prague, Czech Republic*, 2009.

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

[8] E. Lee. Cyber-physical systems-are computing foundations adequate. In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, 2006.

[9] R. Lippmann and K. Ingols. *An annotated review of past papers on attack graphs*. Massachusetts Institute of Technology, Lincoln Laboratory, 2005.

[10] Report: Cyber-physical systems summit. Technical report, National Science Foundation, 2008.

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

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

[13] D. Raymond, R. Marchany, M. Brownfield, and S. Midkiff. Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. *Vehicular Technology, IEEE Transactions on*, 58(1):367–380, 2009.

[14] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2002.

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