Hybrid Attack Graphs for Modeling Cyber-physical Systems

George Louthan
M.S. Candidate

17 Nov 2011

The University of Tulsa
Institute for Information Security

Overview

- Introduction and Motivation
  - Cyber-physical systems
  - Attack graphs
- Background
  - Related formalisms
  - Attack graphs
- Enhanced Discrete Attack Graph
- Hybrid Attack Graph
- Performance
- Conclusions and Future Work

Introduction – Cyber-physical systems (CPS)

- Systems with tight physical-digital-network coupling
- Security threats can come from both sides
- Serious security challenges (Crenshaw and Beyer, 2010):
  - Concentration in safety critical domains
  - **Integration of many third-party or unrelated components**
  - Dependence upon unreliable data collection
  - Pervasiveness

Introduction – Attack graphs

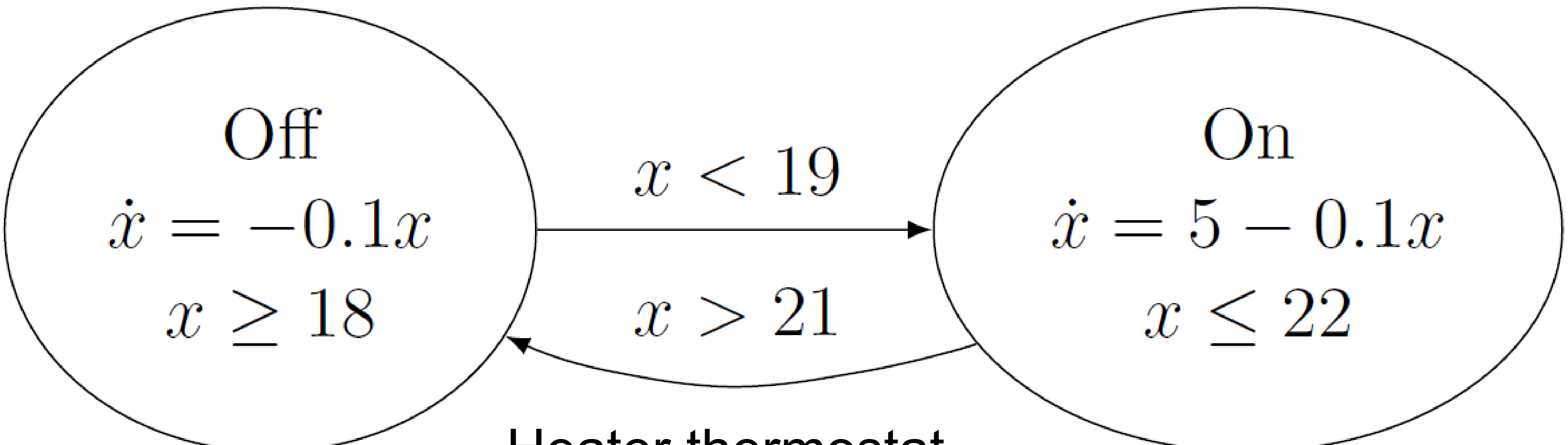
- Discrete domain formalism grounded in graph theory
- Model networks, systems, and attacks
- Excel at highlighting systems' and attacks' interactions

Introduction – Motivation

- Why a formal model for CPS security?
- Safety criticality calls for verification
- Dearth of useful models for the cyber-physical boundary
    - Need a common language for these systems
- Why attack graphs, specifically?
    - System heterogeneity makes them a natural fit

Background – Hybrid systems modeling

- Typical hybrid system model: Hybrid Automata (HA)
- State transition system paired with differential equations
- Operational "modes" and "switches" between them
- Guarded by conditions that govern state variables
- Great for modeling "micro" scale, bad for "macro"

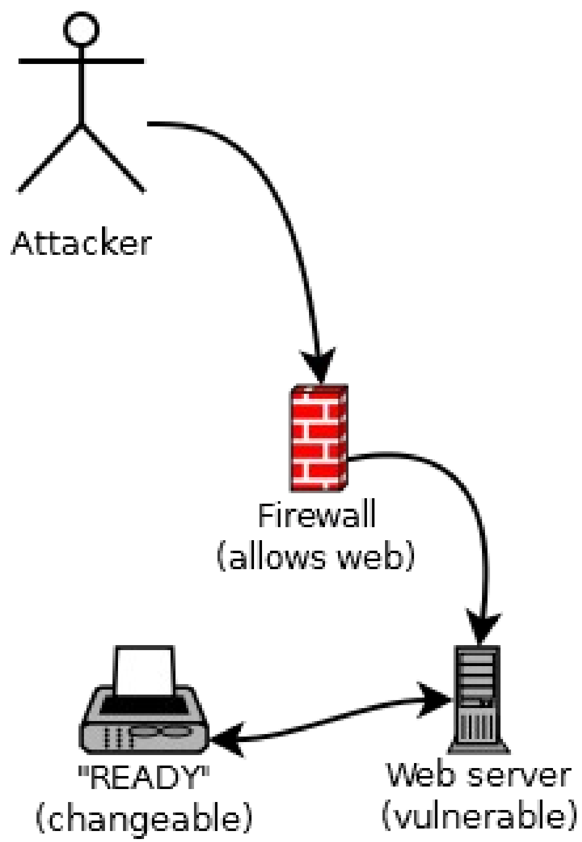


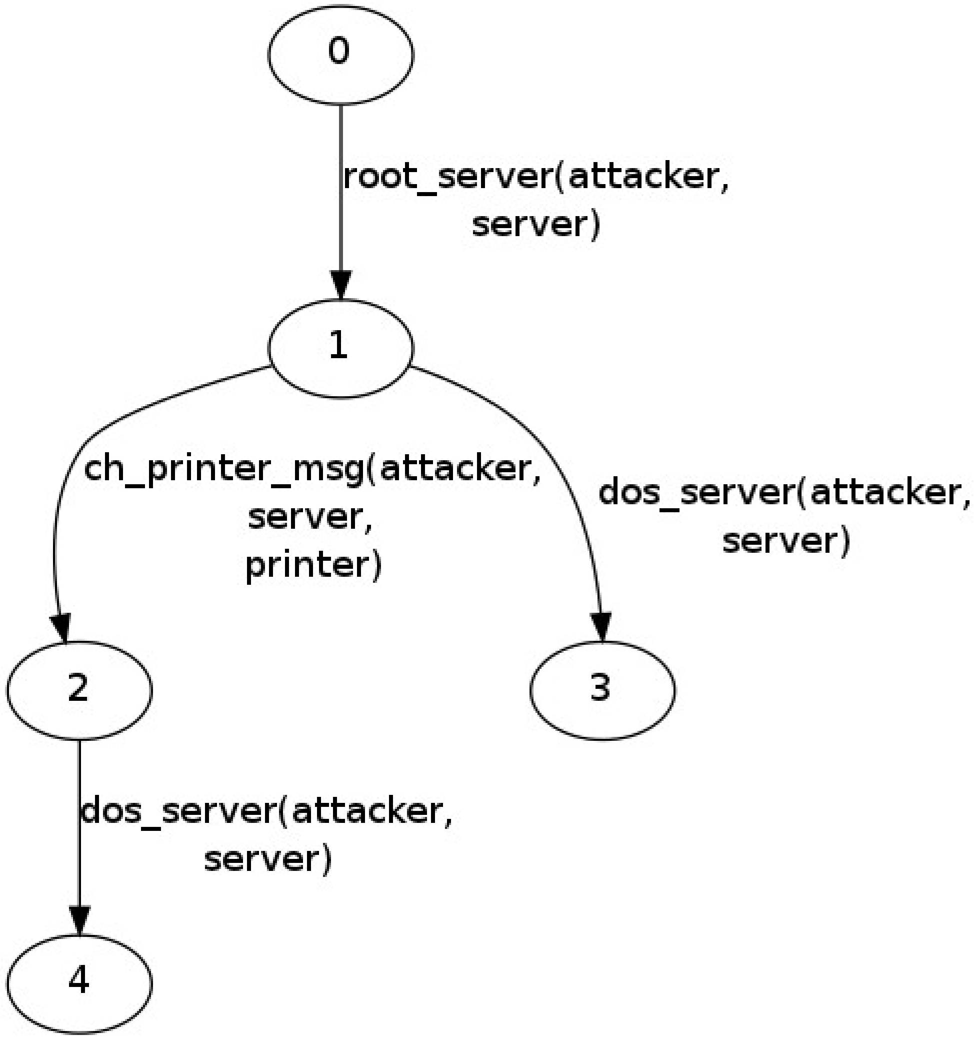
Heater thermostat

Background – Attack Graphs

- Formal model for network attacks
- State transition system
- Automatic generation: matches “exploit patterns” to network objects to create new states
- Network model + exploit patterns + generator = attack graph

Background – Attack Graphs – Example





Background – Attack Graphs – Approaches

- Free form vs. domain driven
  - *Free form – liberal specification semantics, arbitrary keywords, properties, qualities, assets (requires standardized vocabulary)
  - Domain driven – specification uses networking specific concepts (requires heavy lifting in the generator)
- Other mixins:
  - Metrics
  - Likelihood weights
  - Adversary characterization
  - Cost ($$$) of mitigation

Background – Attack Graphs – Domains

Network Model (state)

Facts

- Asset
  - Security principal
- Quality
  - Property of an asset
- Topologies
  - Relation on two assets
  - Directed
- Exploits (free) / Attacks (bound)
  - Function-like
  - Parameters
  - Preconditions
  - Postconditions

```
assets=
 asset1; asset2;
facts=
 quality:asset1,q1=v;
 topology:asset1->asset2,t;
```

```
ex(a1, a2)=
 preconditions:
  topology:a1,q1,v;
 postconditions:
  insert quality:asset1,q2=v;
```

Enhanced Discrete Attack Graphs

- Stepping stone to Hybrid attack graphs (HAGs)
- Contributions
  - Lexicon
    - Access topologies
    - Connection topologies
    - Host status qualities
  - Syntactic improvements
  - Platform facts
  - Global/grouped exploits

Enhanced Discrete Attack Graphs - Lexicon

- Free-form modeling capability requires conventions
- Based upon Common Vulnerability Enumeration and National Vulnerability Database specifications
- NVD/CVE specify useful characteristics of vulnerabilities:
  - “Security Protection” (unauthorized access) types:
    - User access
    - Administrative access
    - Other access (e.g. service account)
  - Access vector:
    - Local
    - Adjacent (same broadcast or collision domain)
    - Network (e.g. Internet)

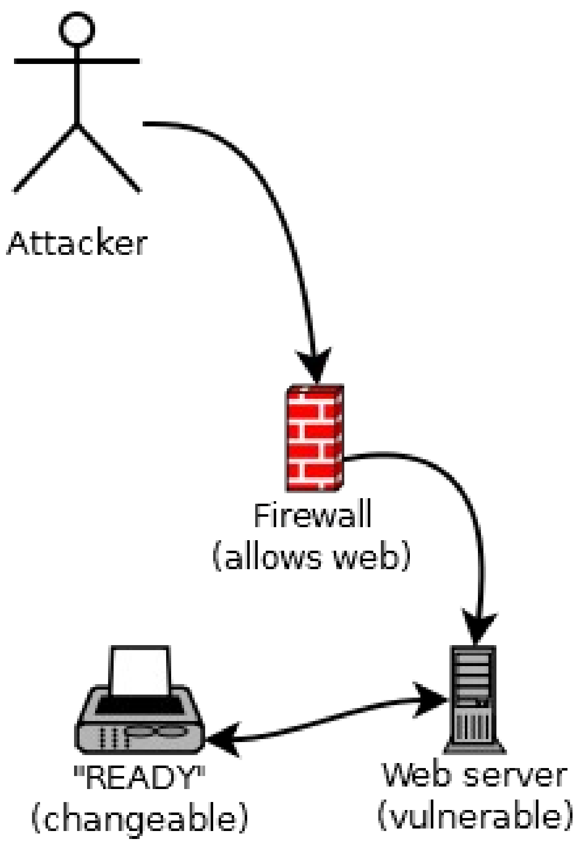
Enhanced Discrete Attack Graphs - Lexicon

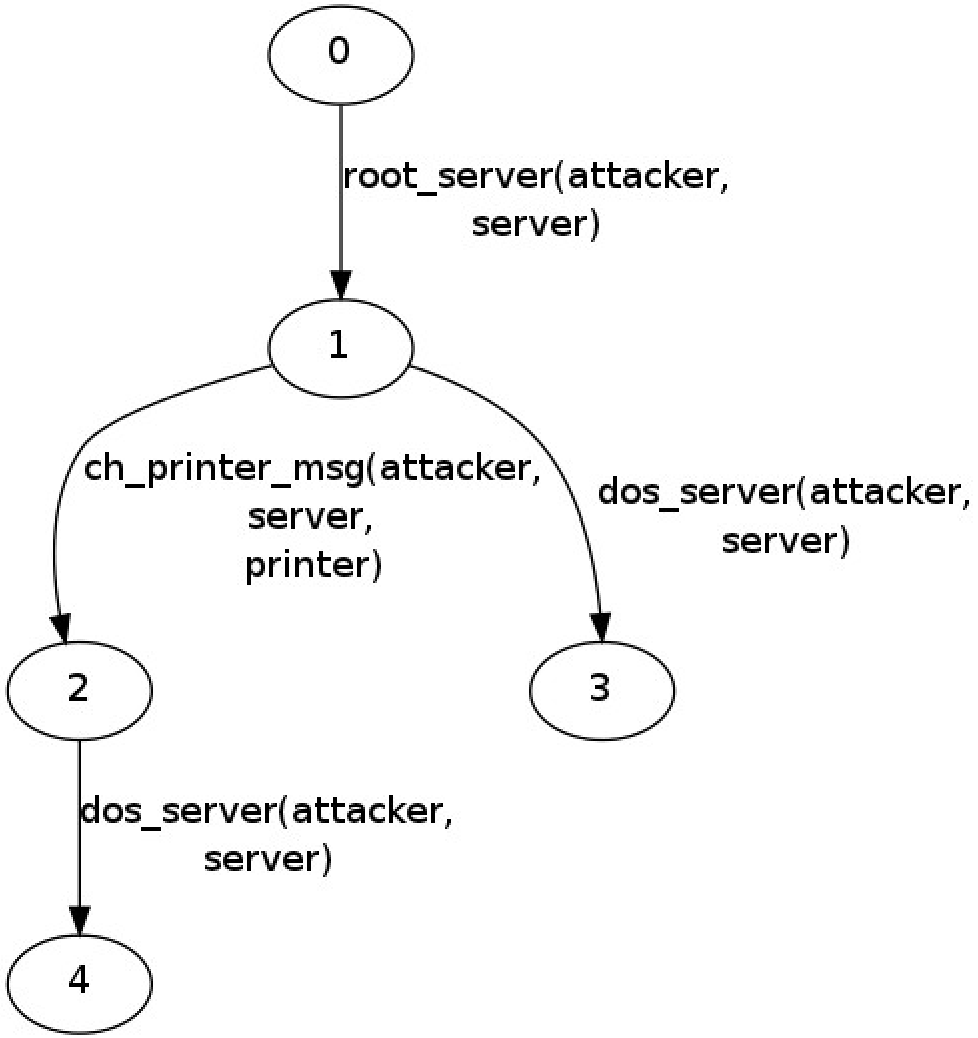
- Status qualities
  - Denote hosts that are up or down
  - Quality name: “status”, values “up” or “down”
- Access topologies
  - Denote (possibly legitimate) trust relationships
  - Topology name: “access_<type>”
  - Type = “user” | “root” | “other” | “other_<name>”
- Connection topologies
  - Denote communication relationships
  - Topology name: “connected_<type>”
  - Type = “adjacent” | “local” | “network_<protocol>”

Enhanced Discrete Attack Graphs - Platforms

- Uses Common Platform Enumeration (CPE)
  - Provides a language for specifying platform hardware, software, version, language, etc.
  - Format: “cpe:/<part>:<vendor>:<version>:<update>:<edition>:<language>”
  - Example: “cpe:/a:adobe:reader:8.1::en”
  - Prefix and wildcard properties apply
- New fact type: platforms (special case of quality)
  - `platform:asset1,cpe:/a:adobe:reader:8.1;`

Enhanced Discrete Attack Graphs – Example





Enhanced Discrete Attack Graphs – Example

NETWORK MODEL

```
network model =
    assets :
        attacker;
        server;
        printer;

    facts:
        topology:attacker->server,connected_network_web;
        topology:server<->printer,connected_local;
        platform:server,cpe:/a::VulnerableWebServer;
        platform:printer,cpe:/h::printer;
        quality:server,status=up;
        quality:printer,ready_message=READY;
.
```

EXPLOIT PATTERNS

```
exploit root_server(a, s)=
    preconditions:
        topology:a->s,connected_network_web;
        platform:s,cpe:/a::VulnerableWebServer;
    postconditions:
        insert topology:a->s,access_admin;
.

exploit ch_printer_msg(a, s, p)=
    preconditions:
        topology:a->s,access_admin;
        topology:s<->p,connected_local;
        platform:p,cpe:/h::printer;
        quality:s,status=up;
    postconditions:
        update quality:p,ready_message=OUT_OF_TONER;
.

exploit dos_server(a, s)=
    preconditions:
        topology:a->s,access_admin;
        quality:s,status=up;
    postconditions:
        update quality:s,status=down;
```

Enhanced Discrete Attack Graphs - Grouping

- It is useful to cause attacks to fire simultaneously
- Aggregate AG edge transition to include multiple attacks
- **Group** exploits
  - `group(group1) exploit ex1(ap) = ...`
  - `group (group1) exploit ex2(ap) = ...`
  - When possible, ex1(a) and ex2(a) use same edge
- **Global** exploits
  - `global exploit ex3(a) = ...`
  - When possible, ex2(a), ex2(b), etc. use same edge
- **Global group** exploits
  - `global group(group2) exploit ex4(ap)`
  - Combines both mechanics

Hybrid Attack Graphs (HAGs)

- AGs excel at modeling disparate components' interactions
  - Recall that this is a principal challenge in CPS
- HAGs extend AGs with real-values, physical reasoning:
  - RFID inventory tag distance to reader
  - Generator temperature or pressure
  - Centrifuge speed

Hybrid Attack Graphs (HAG)

- What does a hybrid attack look like?
  - System placed into harmful **mode** (think HAs)
    - Centrifuge spins too fast
    - Brakes stop working, accelerator stuck
    - Power consumption increased
- Requires *time*
  - Enabled by global/group exploits

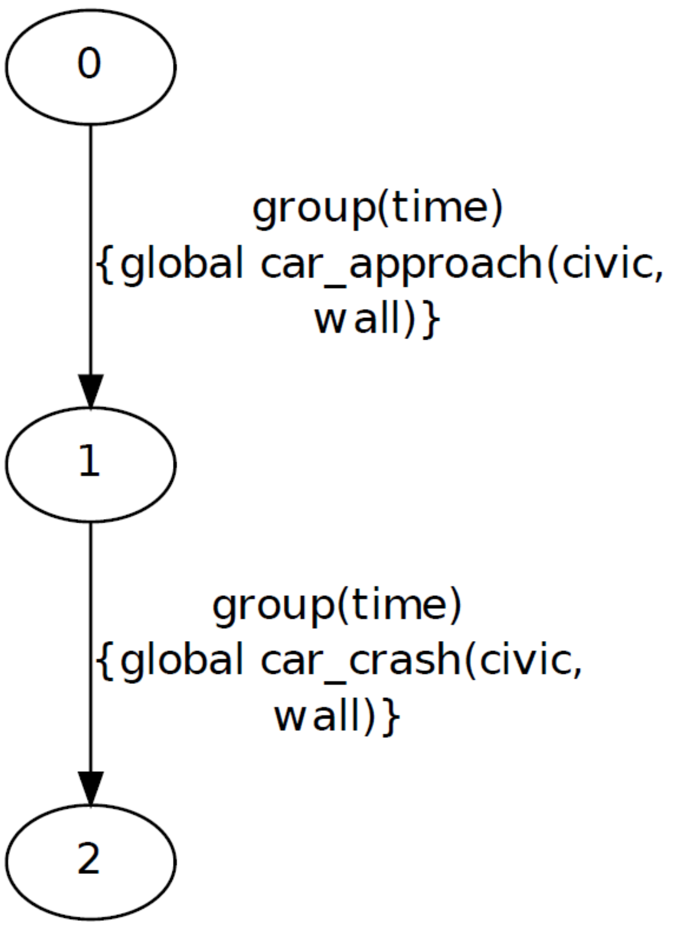
Additions

- Type system
  - Facts are “token” (string) or “real” valued
  - Types denoted by operators
    - Token
      - Assignment: =
      - Relational: =, !=
    - Real
      - Assignment: :=, +=, -=, *=, /=
      - Relational: ==, <>
  - Every test or assignment has an explicit type
- Real (but not token) values for topologies

Additions

- Modeling time
    - Implemented with exploits: `global group(time)`
    - Discretize time into steps of equal size
    - Each time attack is one timestep
- Automotive example

```
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;
.
```



RFID Example

- ISO 18000-7 RFID system used by DOD and DOE for inventory tracking
- Tags are active and battery powered
- Usage includes inventory on nuclear material containers
- Tags have “active” and “sleep” modes
- A “denial of sleep” attack from a rogue or compromised reader causes battery drain

RFID Example

```
network model =
    assets :
        attacker;
        reader;
        tag0;

    facts :
        # Reader:
        platform:reader,cpe:/h::VulnerableReader;
        quality:reader,status=up;

        # Tag:
        platform:tag0,cpe:/h::Tag;
        quality:tag0,status=up;
        quality:tag0,power:=100;
        quality:tag0,mode=sleep;

        # Topologies:
        topology:attacker -> reader,connected_netv
        topology:reader -> tag0,connected_rfid;
.
```

```
exploit own_reader(a, r)=
    preconditions:
        platform:r,cpe:/h::VulnerableReader;
        quality:r,status=up;
        topology:a->r,connected_network;
    postconditions:
        insert topology:a->r,access_admin;
.

exploit deny_sleep(a, r, t)=
    preconditions:
        platform:r,cpe:/h::VulnerableReader;
        quality:r,status=up;

        platform:t,cpe:/h::Tag;
        quality:t,status=up;
        quality:t,mode=sleep;

        topology:a->r,access_admin;
        topology:r->t,connected_rfid;
    postconditions:
        update quality:t,mode=wake;
.
```

Discrete exploits

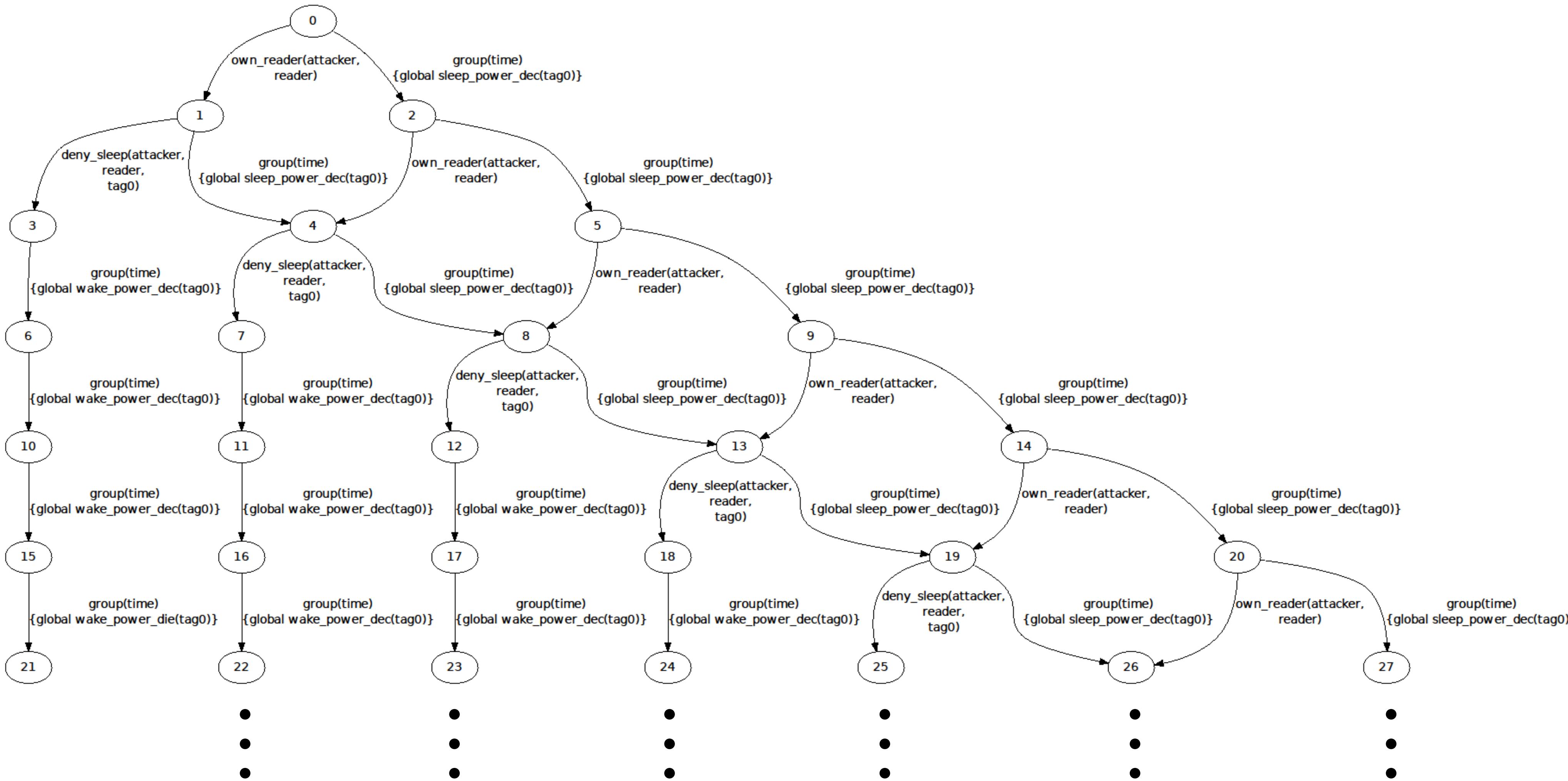
RFID Example

```
global group(time) exploit wake_power_dec(t)=
    preconditions:
        platform:t,cpe:/h::Tag;
        quality:t,status=up;
        quality:t,mode=wake;
        quality:t,power>25;
    postconditions:
        update quality:t,power-=25;
.

global group(time) exploit sleep_power_dec(t)=
    preconditions:
        platform:t,cpe:/h::Tag;
        quality:t,mode=sleep;
        quality:t,status=up;
        quality:t,power>10;
    postconditions:
        update quality:t,power-=10;
.
```

```
global group(time) exploit wake_power_die(t)=
    preconditions:
        platform:t,cpe:/h::Tag;
        quality:t,mode=wake;
        quality:t,status=up;
        quality:t,power<=25;
    postconditions:
        update quality:t,power:=0;
        update quality:t,status=down;
.

global group(time) exploit sleep_power_die(t)=
    preconditions:
        platform:t,cpe:/h::Tag;
        quality:t,mode=sleep;
        quality:t,status=up;
        quality:t,power<=10;
    postconditions:
        update quality:t,power:=0;
        update quality:t,status=down;
.
```

Time exploits

RFID Example



Performance

Domain	Symbol
Facts	$F = \lvert \mathcal{Q} + \mathcal{T} \rvert$
Fact names	f
Exploits	$E = \lvert \mathcal{E} \rvert$
Depth	d
Assets	$A = \lvert \mathcal{A} \rvert$
Qualities	$Q = \lvert \mathcal{Q} \rvert$
Topologies	$T = \lvert \mathcal{T} \rvert$
Most preconditions in any exploit	p
Most postconditions in any exploit	P
Most parameters in any exploit	a
Possible asset bindings	B

Build attack graph $O(1)$

$$O\left(\left(\left(\frac{A!}{(A-a)!}\right)E\right)^d\right)$$

- **Load initial state** $O(F)$
- **Load exploits** $O(E)$
- **Generate attack graph (recursive)** $O(d)$
  - **For each analysis state** $O(states)$
    - **Copy network model** $O(A) + O(F)$
    - **Get valid attacks** $O(1)$
      - **For attack in attack bindings** $O(BE)$
        - **Validate attack** $O(p)$, which is technically bounded by $O(A^2) \cdot O(F)$ but is in practice quite small
        - **Process groups and globals** $O(1)$
    - **For each valid attack** $O(BE)$
      - **Get successor state** $O(1)$
        - **Build network model** $O(A) + O(F)$
        - **Add postconditions** $O(P)$, which is technically bounded by $O(A^2F)$ but is in practice quite small.

Performance

Domain	Symbol
Facts	$F = \|\mathcal{Q} + \mathcal{T}\|$
Fact names	f
Exploits	$E = \|\mathcal{E}\|$
Depth	d
Assets	$A = \|\mathcal{A}\|$
Qualities	$Q = \|\mathcal{Q}\|$
Topologies	$T = \|\mathcal{T}\|$
Most preconditions in any exploit	p
Most postconditions in any exploit	P
Most parameters in any exploit	a
Possible asset bindings	B

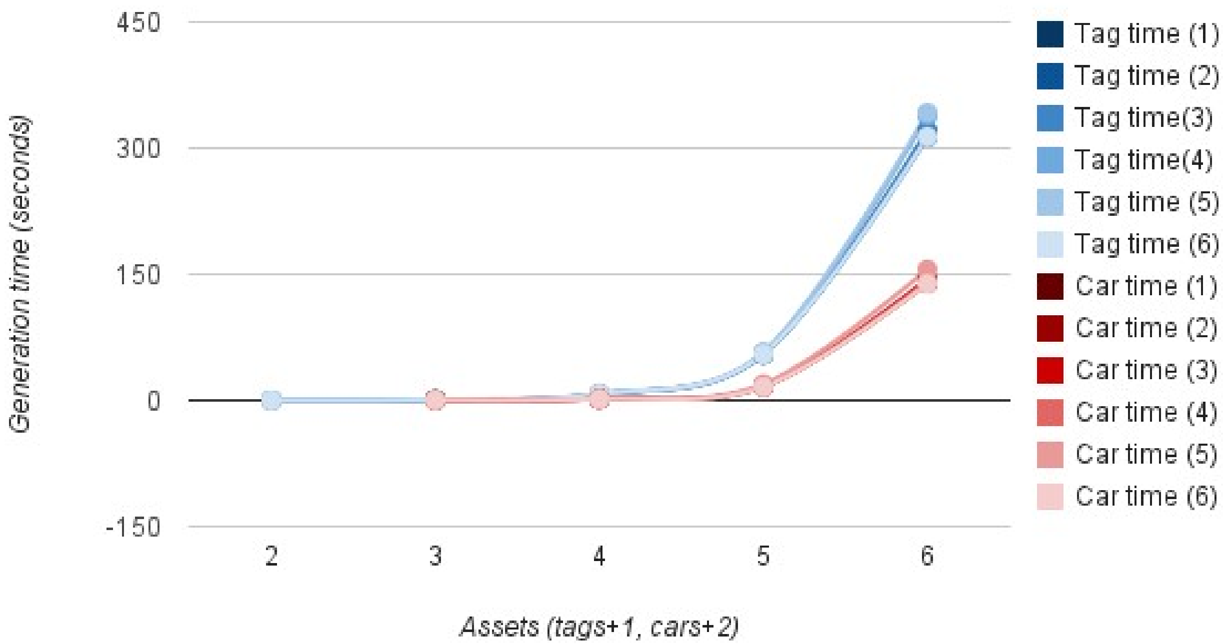
$$O((A^A E)^d((A + A^2 f) + A^A E A + A^A E(A + A^2 f + A)))$$

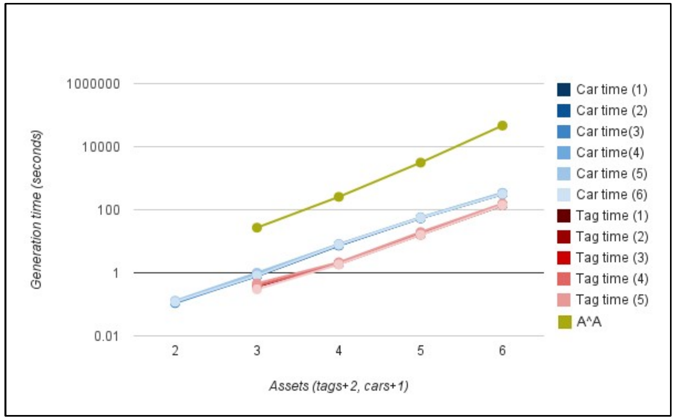
$$= O(f E A^{Ad+A+2})$$

$$= O(E^{d+1} f A^{A(d+1)+2})$$

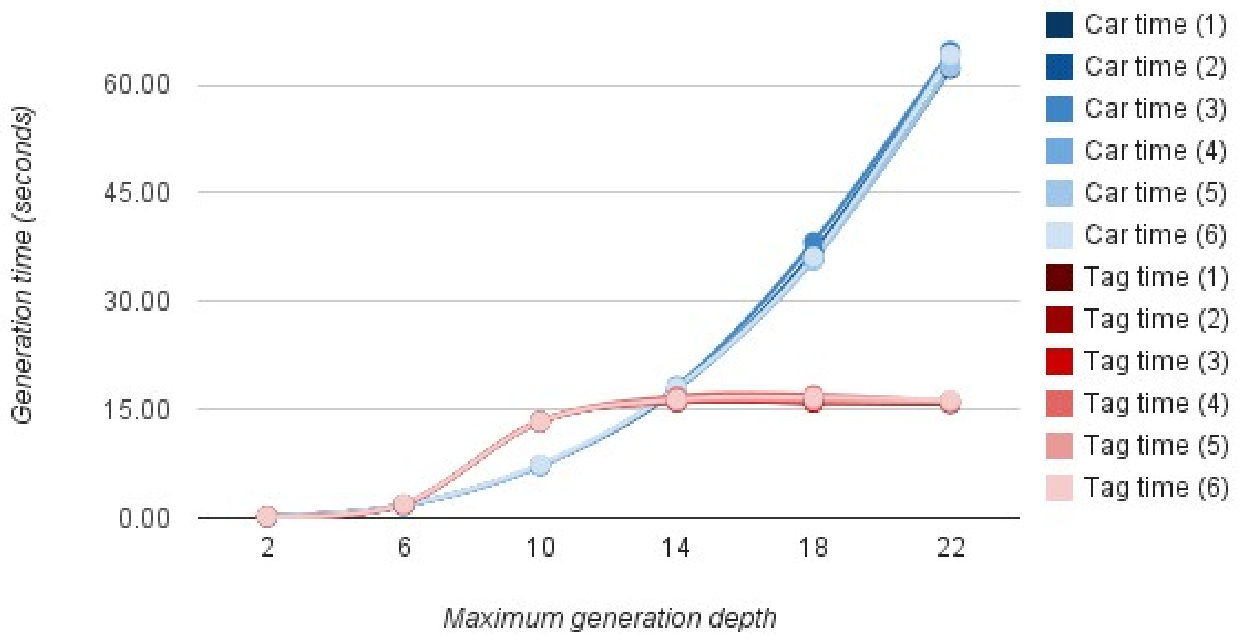
$$\approx O(E^{d+1} f A!^d) \text{ (in a mild abuse of notation)}$$

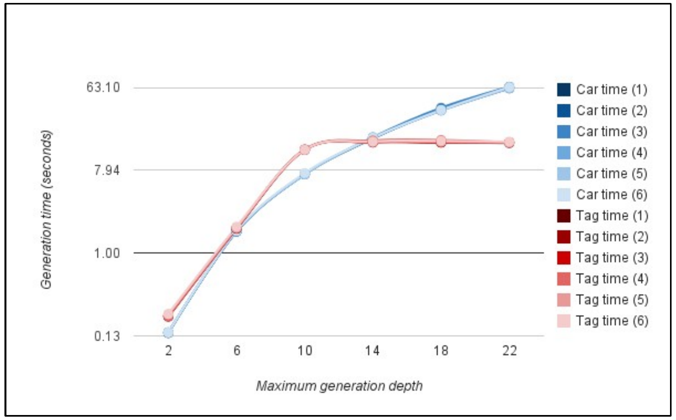
Performance – Number of assets





Performance – Generation depth





Performance Summary

- Execution time scales factorially with input assets and exponentially with maximum depth
- Falls well short of the state of the art
  - This is expected
  - This is fine

Conclusions – Summary

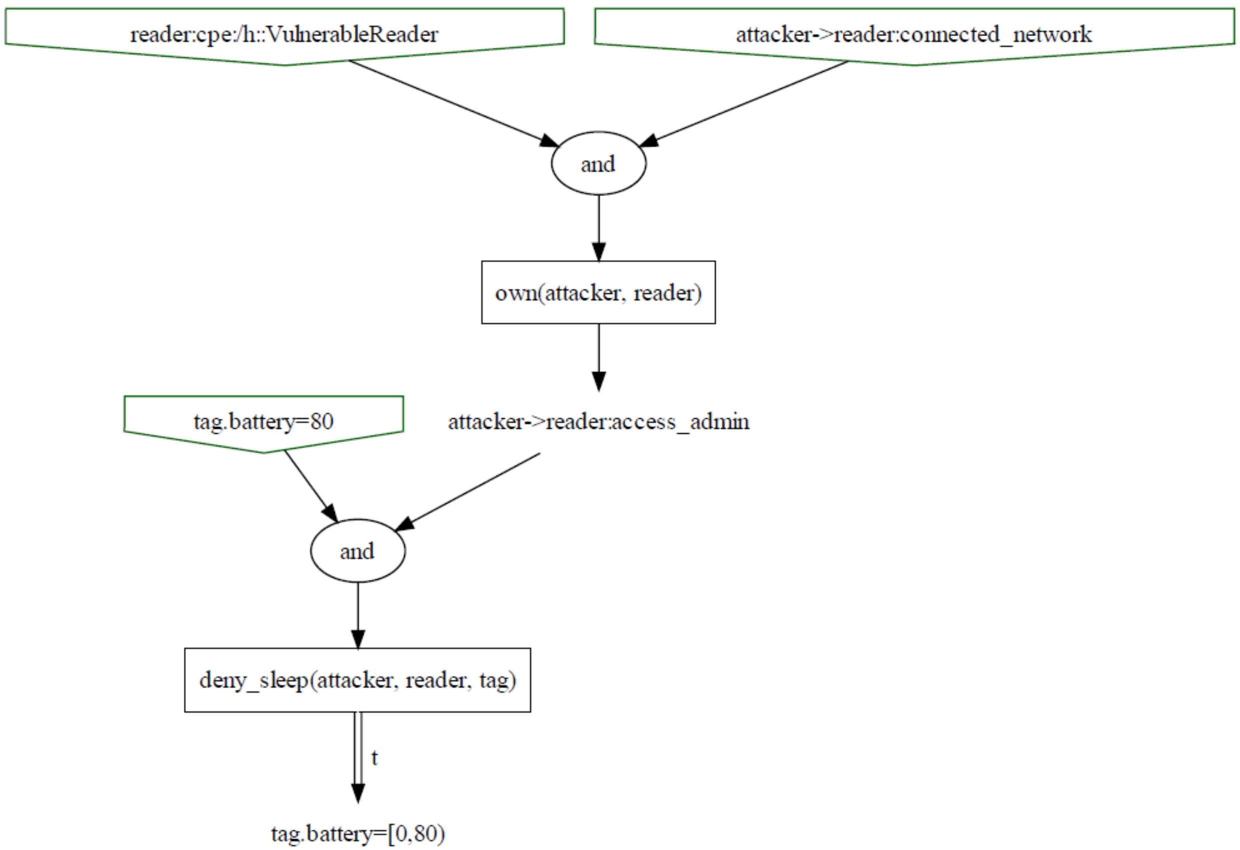
- Enhanced discrete attack graph with applications to information systems (IS)
- Expanded attack graph with applications to CPS, possibly to IS as well
- Faithfully model well-behaved subset of hybrid systems
- Falls short of the state of the art in performance

Conclusions - Challenges

- Performance
  - HAGs make little changes to algorithmically expensive portions of the generation process, however
- Cognitive scalability
  - Even more marked in time driven HAGs
- Model acquisition
- Less serious:
  - Separation of concerns (exploits vs. network)
  - Wildcard and pivot exploits
  - Complex preconditions
  - Variable hybrid postconditions (HA applications)

Future Work

- Hybrid attack dependency graphs



Acknowledgments

- Committee
  - Dr. John Hale, Dr. Mauricio Papa, Dr. Peter Hawrylak
- Collaborators
  - Chris Hartney, Matt Young, Carsten Mueller, Aleks Kissinger, Zach Harbort, Jordan Sanderson, Phoebe Hardwicke
- Funding (DARPA)

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.