International Conference on Dependable Systems & Networks: Anchorage, Alaska, June 24-27 2008

Automatic Security Assessment of Critical Cyber-Infrastructures [*]

Zahid Anwar, Ravinder Shankesi, Roy. H. Campbell
{*anwar,rshanke2,rhc*}@*uiuc.edu*
University of Illinois at Urbana-Champaign

Abstract

This research investigates the automation of security assessment of the static and dynamic properties of cyberinfrastructures, with emphasis on the electrical power grid. We describe a network model representing the static elements of a cyberinfrastructure including devices, services, network connectivity, vulnerabilities, and access controls. The dynamic elements include workflow models of the operating procedures, processes and the state of a working power grid. We introduce a toolkit that with a little manual assistance can automatically generate these models from specifications, continuously update attributes from online event aggregators, and perform security assessment. The assessment reveals whether observed anomalies about the system could indicate possible security problems and permit dynamic ranking of alternative recovery procedures to minimize the total risk. We motivate the use of the tool-chain by showing an example scenario where the recovery procedure recommended to minimize security risk depends on the current state of system as well as the network topology.

1 Introduction

Computerized control systems, also referred to as Supervisory Control and Data Acquisition (SCADA) systems, have become vital in the modern world. SCADA is deployed to control water supply, telecommunications as well as electricity generation and distribution. In this paper we focus on SCADA systems for the electrical power grid. These systems typically use off-the-shelf computing and networking components, for connecting to enterprise networks or the Internet, making them vulnerable to well-known cyber attacks. Furthermore, knowledgeable and inside attackers can use the properties of the power grid and its operating procedures to cause cascading failures, power blackouts or damage difficult to replace vital resources such as high-power transformers. The security of SCADA systems made headlines [11] recently when researchers at the Department of Energy's Idaho lab launched an experimental cyber attack on an electrical power plant causing a generator to self-destruct. While the details of this particular attack are not explained in depth, what is clear is that researchers were able to remotely hack into the SCADA network and change its configuration to cause significant damage to the generator. A comprehensive report compiled by the Industrial Security Incident Database (ISID) [1], shows an alarming increase in the numbers of security attacks on cyber infrastructures in recent years, with externally generated incidents accounting for 70% of all events between 2001 and 2003. The Slammer Worm infiltration of an Ohio nuclear plant [7] and the Australian sewage spill incident [20] in 2000 are two recent examples. In the latter case an attacker connected through a wireless network used to control sensors for a sewage treatment plant in Queensland, taking control of the main system to drain raw sewage into many of the parks and lakes.

Throughout this paper we model SCADA and Enterprise networks in predicate-logic (henceforth called a *network model*) which consists of a set of devices, services, operating systems, network connections, known vulnerabilities as well as their attributes. The network model is used to generate attack graphs for various SCADA devices to determine vulnerability to external attackers. While evaluating these attack graphs, we compute the *security risk* for each device depending on both the severity of the isolated vulnerabilities of various nodes, on the path to that device, as well as the topology of the paths. Our device security risk is formalized as a lattice whose partial order function depends on the type of vulnerability and the calculated severity of the vulnerability (*i.e.*, an execution control vulnerability is rated higher than a denial of service vulnerability and the severity of a vulnerability may be valued higher or lower depending on its exploitability). The device security risk calculated in the network model is used as an input for our second model explained below.

The second model, called the *workflow model*, describes the various operating procedures, as workflows encoded in rewriting logic. Operational procedures are usually recovery or maintenance activities that the operators follow in the

[*]This work was funded by the UIUC TCIP Project NSF CNS 05-24695.

system, for instance, to recover from a failed component or deal with some contingency. These recovery procedures are made up of a ordered set of tasks that enable or disable SCADA network devices (for instance, a task to allow selection of a backup transformer from a list of idle transformers in a substation). Tasks can be fulfilled in various ways, allowing operators a choice of strategies to perform a particular function. The security *risk* of a particular recovery procedure is calculated by aggregating the risks derived from all the vulnerabilities of each device used in the recovery procedure as obtained from the network model. Finally, the security model presents its evaluation of the possible recovery procedure options to an operator along with their security risks.

We have developed a tool-chain that semi-automates the generation of the network and workflow models as well keeps the attributes up-to-date using on-line SCADA event aggregators. We demonstrate its feasibility to find complicated attacks on our SCADA test-bed that mimics a real SCADA substation. We describe how our approach is scalable, although further automation is clearly possible.

The remainder of this paper is organized as follows: Section 2 provides related work on existing SCADA security models. Section 3 gives an overview of SCADA for Power Systems and rewriting logic. Section 4 outlines the design of our security model and Section 5 shows our implementation via a tool chain based on Prolog, Yet Another Workflow Language(YAWL) [23] and Maude [5]. Section 6 shows and evaluates a realistic workflow advisories scenario. We conclude the paper in Section 7.

2 Related Work

Our research benefits from related work on attack trees, and quantization of security for large-scale safety critical systems. One work on SCADA attack trees [4] describes the application of attack trees to the common MODBUS SCADA protocol with the goal of identifying security vulnerabilities inherent in the specification and in typical deployments. Another interesting use of SCADA attack trees [22] evaluates security improvements based on countermeasure types and password policy enforcement on tree leaves. An optimization problem is formulated to determine pivotal leaves in the tree for security improvement.

Jajodia and Noel have used automated attack graph generation and processing techniques using vulnerability scanning tools like Nessus [13] for aiding sensor placement for monitoring attack paths to critical cyber assets [12]. Researchers have also proposed various methodologies such as compromise graphs [9] and Markov Chains [14] for obtaining a quantitative measurement of the risk reduction achieved when a control system is modified with the intent to improve cyber security defense.

The CORAS [3] project supports methodologies for risk analysis of security-critical systems by modeling threats to a system as unwanted features of the system in question. Users model a system and its associated threats using UML diagrams and XML schemas allowing exchange of risk assessment data in a formalizied and standardized language. We improve upon their idea of using UML by employing the standard descriptive language based on Common Information Models (CIM) [8] to automate the generation of our security models. CIM, an object-oriented cyber-infrastructure modeling language developed by the Electric Power Research Institute (EPRI) is better suited for modeling electrical utility enterprises.

While security assessment of large distributed systems and the use of attack graphs models to find network vulnerabilities is a fairly mature area, we did not find much work in the automated generation of these models especially for the cyber-infrastructures domain. Moreover there is little work, if any, in using the security vulnerabilities calculated for network elements as inputs to finding risks in operating procedures and providing advisories.

3 Background

This paper does not assume that the reader has a power engineering background and explains the key power terms used throughout the paper in this section. It also includes the threat model we consider for the power system in question. Horn-clause logic and rewriting logic have been used for the formal analysis and while we assume the reader is reasonably familiar with the former we give a brief explanation of the latter here as well.

3.1 A Power System

In a conventional electric power system, energy from fossil fuels or falling water is harnessed to generate steam to drive power turbines that produce electricity, which is then transmitted and distributed to the end user. There are a variety of SCADA controls used throughout the process including controls for turbine, burner, and switching. Focusing on just the switching control; a typical power system will have a large number of switches that affect the way power is routed and distributed within various components. These switches are often controlled remotely through SCADA (but can also be turned on or off manually). Figure 1 shows how a power distribution system can be viewed as a network of electric lines connected via switching devices and fed via circuit-breakers. The supply of power to consumer devices located on the lines is controlled by the circuit-breakers which supply power if and only if (iff) closed, and switching devices that stop power propagation iff open. Mostly lines have a meshable structure exploited radially and the positions of the devices are set so that the paths taken by the power of each circuit-breaker form a tree called a feeder. The root of a feeder is a circuit-breaker, and its leaves are whatever switching devices downstream happen to be open at the time.

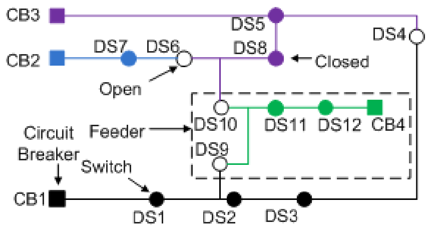


Figure 1. Power Distribution System

Power lines are often subject to faults (short circuits) that are mainly due to bad weather conditions and lightning. Upon occurrence of a fault, the circuit-breaker feeding the faulty line opens in order to protect the rest of its feeder from damaging overloads. For instance, if a fault occurs on the line between DS1 and DS2, CB1 will open leaving all consumers located on that feeder without power. Simply re-closing the circuit-breaker will not help but instead, SCADA devices (called actuators) controlling the switches need to be used to locate faulty lines and then reconfigure the network to isolate them and restore the supply to the non-faulty lines. Remote-controlled actuators sense and change the position of switches and report sensing the presence of faults. Changing the status of switching devices in a substation allows some interesting attack scenarios from an intruder's point of view. A denial of service attack on the actuator or its controlling SCADA device such as a Programmable Logic Controller (PLC) or relay would lead to a failure to report the proper state in time (and might require manual intervention). Even more seriously, a buffer-overflow in a networked device (allowing execution privileges) can allow an attacker to black-out a feeder or overload a transformer. The latter is a very serious attack as transformers are expensive and hard to replace.

3.2 Rewriting Logic

We specify the operating procedures as workflows in rewriting logic. In general, a concurrent system can be specified in rewriting logic [10] as the theory $\mathcal{R} = (\Sigma, E, R)$ where (Σ, E) is the *order-sorted equational theory* such that:

- The signature Σ specifies the sorts[1], a sub-sort relation, constants and function symbols. The terms T_Σ and $T_\Sigma(X)$ denote,the terms the set of ground Σ-terms and the set of Σ-terms over variables in X.

[1]A sort can be informally thought of as the type of a term.

- The equations in E are of the form

$$(\forall X)\ u = \ v \textbf{ if } C$$

where u, v are of the same sort and the (possibly null) condition C is a conjunction of unquantified Σ-equations involving variables (only) in X. We say the Σ-algebra A satisfies the equation $(\forall X)\ u = v$ **if** C iff for each assignment $a : X \to A$, $a(u) = a(v)$

Intuitively, the theory (Σ, E) defines the states of the system and has the initial model $T_{\Sigma|E}$. The *dynamics* are described by the *rewrite rules* R that specify concurrent transitions that can occur in the system and that can be applied *modulo* the equations in E.

Rewrite rules are of the form [2]
$(\forall X)\ u \to v \quad$ **if** $\bigwedge_i u_i = v_i$
and describe a transition from the term t to term t'. To apply the rewrite rule to the term t, we find a subterm of t which is an instance of u under some substitution σ. We substitute u in t by v, only if all the conditions hold i.e., $\forall i : \sigma(u_i) = \sigma(v_i)$. Note that multiple rewrite rules may be applicable to a given term.

Given a rewrite-theory (Σ, E, R) we can define the transition relation $\to$ over the states (given by the terms in the algebra $T_{\Sigma|E}$) by using the one-step rewrite rule in R. We can label the transition system given by $(T_{\Sigma|E}, \to)$ by using predicates defined using equations P[3] that associate a term in (Σ, E) to a proposition. Therefore, given a rewrite theory, (and appropriate labeling) we can define the Kripke-structure that describes the transition system. The extension from a rewriting logic to the Kripke structures on which the LTL model checking works is described in greater detail in [6]. Therefore, given a system described using term-rewriting logic, we can verify if it satisfies a given LTL property by using LTL model checking.

Rewrite theories are *executable* (under reasonable assumptions over E, R). In this work, we use Maude [6] to implement our *workflow model*. Maude supports LTL model checking by using an on-the-fly model checker.

4 Security Model

Our formal model is composed of two parts. A *network model* captures the static parts of a SCADA system comprising the network topology, devices, services, connectivity and vulnerabilities (known software exploits). A *workflow model* captures the dynamic parts such as maintenance, recovery activities involved and their ordering and relationships.

[2]Note that, in general, rewrite rules can be slightly more complicated, but we describe a simpler notation adequate for this paper.
[3]We require that $(\Sigma, E \cup P)$ be a *protecting* extension of (Σ, E).

4.1 Network Model, $\mathcal{N}$

The network model $\mathcal{N}$ represents the SCADA network as two types of graphs: a dependency graph G and logical attack graphs G'. The dependency graph is given as $G = (D, E)$, where D is the set of all devices and $E \subseteq D \times D$ is the set of edges between two physically connected devices. A set of functions gives the mapping between devices and their attributes such as services, privilege levels and vulnerabilities (for further details on how we formalize network dependencies see [2]).

G is modeled as primitive facts in first order predicate logic. The security risk of a device is dependent upon an attacker's ability to exploit a vulnerability V on that device or on a device from which it is reachable. Attack graphs are a well known technique [16, 15, 19] that represents a chain of exploits as a *path*, where each exploit in the chain lays the groundwork for subsequent exploits. The pioneering work [19] in this area used model checkers to identify explicit attack sequences. However, this approach suffers from scalability issues because the number of such sequences grows exponentially with the product of the number of vulnerabilities and devices. Later work [15] proposes a new logic based approach where each node in the graph is a logical statement and edges are causality relations between network configurations and attacker privileges. This results in the attack graph size being polynomial in the size of the network. We reuse their technique for generating our attack graphs.

We use two algorithms *ConstructAttackGraph* and *EvaluateAttackRisk* that are executed sequentially for each device whose security risk is being evaluated. The first algorithm constructs a graph depicting all possible ways an attacker could effect a device's safety (e.g. compromise of availability, integrity). The *EvaluateAttackRisk* algorithm then uses this output graph as input and calculates the overall security risk for the device.

ConstructAttackGraph essentially records a successful Prolog derivation (an implementation of the backward chaining algorithm in horn clause logic) as an attack graph G'. Similar to [15] the logical attack graph G' is a tuple $(N_r, N_p, N_d, E', \tau, \gamma)$ where N_r, N_p and N_d are three sets of disjoint nodes in the graph, $E' \subset (N_r \times (N_p \cup N_d)) \cup (N_d \times N_r))$, τ is a mapping from a node to its label, and $\gamma \in N_d$ is the attacker goal needed to perform the exploit r_l on device D[4]. N_r, N_p and N_d are the sets of rule nodes, primitive fact nodes and derived fact nodes respectively. Primitive fact nodes N_p were described earlier in the construction of G. N_r represent predicate rules that describe conditions on N_p to form derived nodes N_d. The root node of the attack graph is the goal we are trying to satisfy e.g. 'is a device vulnerable to DoS' while the primitive facts, such as knowledge about exploits, form the leaves.

[4]for e.g., *codeExecute*,*DoS* to exploit integrity,availability resp.

To find the accumulative security risk D_M associated with the root device node we use the following heuristics:

- H1: Security Risk decreases if the 'length of the paths' leading to the victim increases following the analogy that the difficulty accumulated in reaching a target is proportional to the number of locks to be opened.
- H2: Security Risk increases if the 'number of paths' leading to the target is large. The attacker can use the different paths simultaneously to break different locks on each path.

We can translate these heuristics into a graph traversal algorithm (see algorithm 1) that evaluates the security risk. This algorithm is essentially a variant of the recursive depth-first search algorithm over a directed acyclic graph. Intermediate nodes evaluate their security risk by recursively calling *EvaluateAttackRisk* on their children and returning a product of their own risk and that of their children. Individual exploit likelihoods are available from vulnerability databases such as [18, 21]). We represent the exploitability as a real-number between 0 and 1 where a higher number indicates that a device is more vulnerable. An individual node's exploitability depends on the set of known individual vulnerabilities for that node as well as the number of paths that reach that node. Firstly, a device's individual exploitability with n possible individual vulnerabilities is calculated as $1 - \Pi_n(1 - p_n)$ were p_n corresponds to the exploitability values of individual vulnerabilities. Secondly, for a vulnerable device with individual expoitability p_d and i possible paths reaching it each with exploitability p_i, its cumulative exploitability is calculated as $1 - \Pi_i(1 - p_d \times p_i)$. This ensures that longer paths will decrease attack risk.

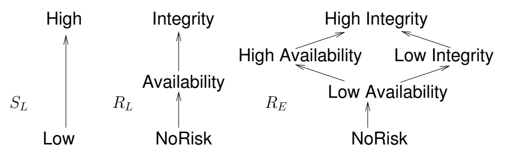
```
/*initialize Risk=1 for all nodes*/
double procedure $EvaluateAttackRisk(V)$
  if $Visited(V)$ then
    return $Risk(V)$
  end if
  $markAsVisited(V)$
  /*A rule node's individual exploitability: calculated from all vulnerabilities */
  if $isRuleNode(V)$ then
    for EACH $I \in AdjacentPrimitiveNodesSet(V)$ do
      $Risk(V) \leftarrow Risk(V) \times (1 - Exploitability(I))$
    end for
    $Risk(V) \leftarrow (1 - Risk(V))$
  end if
  /*If a Leaf node then just return your self risk*/
  if $isLeafNode(V)$ then
    return $Risk(V)$
  end if
  /*If an intermediate node then recursively evaluate risks of children*/
  $childRsk = 1$
  for EACH $I \in ChildSet(V)$ do
    $childRsk \leftarrow (1 - EvaluateAttackRisk(I) \times Risk(V)) \times childRsk$
  end for
  $Risk(V) \leftarrow (1 - childRsk)$
  return $Risk(V)$
```

Algorithm 1: Evaluates the risk associated with an input attack graph

Security Risk: The algorithms *ConstructAttackGraph*, *EvaluateAttackRisk* together give, for each vulnerable device D, the tuple (r_l, D_M), where $r_l \in R_L$ and $(R_L, \leq)$

is the security risk lattice which characterizes the *kind* of vulnerability that device D has, and $D_M \in \mathbb{R}$ s.t($0 \leq D_M \leq 1$) is the *severity* of the vulnerability (the cumulative exploitability). We use D_M to assign a severity label to the original risk as follows $Sev : \mathbb{R} \rightarrow S_L$ where $(S_L, <)$ is the set of labels with a total order[5].

Given the risk lattice R_L and the severity label S_L, we define the new extended risk-lattice R_E as follows $R_E \subset \{(R_L \times S_L)\}$. The new security lattice is defined by the partial order operator $\leq_E$ such that (given $R, R' \in R_L$ and $S, S' \in S_L$ and $\forall R \in R_L : min(R_L) \leq R$):
$(R, S) = (R, S')$ if $(R = \min(R_L))$
$(R, S) \leq_E (R', S)$ if $(R \leq R')$
$(R, S) \leq_E (R, S')$ if $(S < S')(and)(R \neq \min(R_L))$
$(R, S) \leq_E (R', S')$ if $(R \leq R')$ and $(S < S')$
The new *security-risk* for each vulnerability is now given by $(r \times Sev(D_M)) \in R_E$. For instance, the final risk-lattice for a system with $\{NoRisk, Availability, Integrity\}$ as the vulnerabilities and $\{Low, High\}$ as the severity can be depicted as follows.



4.2 Workflow Model $\mathcal{W}$

We formalize the notion of recovery and operating procedures in the form of workflows. There are many different workflow description languages and our model of workflows is a subset of YAWL's basic control flow patterns. One distinction is that we add the notion of "actions" to be performed by a task. Here we have a list of actions that can be performed, and each of which has a security risk. The next task that can be fired as well as the actions that are chosen at any given task can be *context-sensitive*. The *context* consists of the list of (task,action) pairs performed in the workflow so far [6]. At any given task, the workflow execution makes a non-deterministic choice between the list of actions available at that task. Although this can be modelled in terms of basic workflow primitives, our method of allowing non-deterministic choice at a task allows us to use a "generic" workflow description for various different power-grid environments. The individual "environment-specific" choices that can occur at any given task can be modeled as actions possible at that task.

Formally, we define the workflow as the tuple, $\mathcal{W} = (T, C, F, A_{id}, R_E, \leq, A, R, S_t, J_t)$, where each element in the tuple is defined in Table 1

[5]For e.g., $S_L = \{High, Low\}$ such that $(Low < High)$
[6]We do not show the context in the model to simplify the description.

t

Element	Definition
T	set of Tasks
C	set of Conditions.
$F \subseteq (T \times C) \cup (C \times T)$	transition between Tasks and Conditions.
A_{id}	set of all actions possible.
R_E	set of security risks (as defined in Section 4.1) and $\leq$ is the partial-order over elements in R_E, such that $(R_E, \leq)$ is a lattice.
$A : T \rightarrow \mathbb{P}(A_{id})$	is the set of actions associated with any given task.
$R : A_{id} \rightarrow R_E$	maps a risk with each action.
$S_t : T \rightarrow \{AND, XOR\}$	is the split condition[7].
$J_t : T \rightarrow \{AND, XOR\}$	is the join condition.

Table 1. Workflow Definition

The semantics of the workflow are defined in terms of a transition system over workflow states. A *workflow state* for a given workflow $\mathcal{W}$ is defined by the tuple $W_s = (Tk, R_a, H)$ where:

- $Tk \subseteq \{(T \times Tk_s) \cup (C)\}$ is the set of tokens present at any of the tasks or conditions. A token at a task $t \in T$ can be in one of two states $Tk_s = \{ENABLED, FINISHED\}$.
- $R_a \in R_E$ refers to the accumulated risk over all the actions performed at all the finished tasks, i.e., $R_a = \cup_a(R(a)), \forall a \in H$.
- $H \subseteq \{(T \times A_{id})\}$ stores the set of actions performed at all the finished tasks.

The transition relation between two consecutive workflow states in the system depends on the workflow transition relation F and the split, join conditions S_t, J_t. We define the state transition relation $\delta : W_s \rightarrow W_s$ over the workflow states W_s corresponding to a workflow $\mathcal{W}$ as follows:

Join Processing

- $\delta(\{Pred(T) \cup Tk, R_a, H\}) = \{(T, ENABLED) \cup Tk, R_a, H\}$ if $J_t(T) = AND$, where $Pred(t) = \{c \mid (c, t) \in F\}$ and F is the flow relation in workflow $\mathcal{W}$. $Pred(T)$ defines the set of all predecessor condition nodes for task T in the workflow $\mathcal{W}$.
- $\delta(\{T_p \cup Tk, R_a, H\}) = \{(T, ENABLED) \cup Tk, R_a, H\}$ if $J_t(T) = XOR$, where $T_p \in Pred(T)$ for task T in the workflow $\mathcal{W}$.

Task Processing

- $\delta(\{(T, ENABLED) \cup Tk, R_a, H\}) = \{(T, FINISHED) \cup Tk, R_a + R(A_i), (T, A_i) \cup H\}$ where $(A_i) \in A(T)$ and $+$ is the *least upper bound* function for the risk-lattice $(RL, \leq)$ of the workflow $\mathcal{W}$.

Split Processing

- $\delta(\{(T, FINISHED) \cup Tk, R_a, H\}) = \{Succ(T) \cup Tk, R_a, H\}$ if $S_t(T) = AND$, where $Succ(t) = \{c \mid (t, c) \in F\}$. $Succ(t)$ defines the set of all successor condition nodes for task t in the workflow $\mathcal{W}$.
- $\delta(\{(T, FINISHED) \cup Tk, R_a, H\}) = \{T_s \cup Tk, R_a, H\}$ if $S_t(T) = XOR$, where $T_s \in Succ(T)$ for task T in the workflow $\mathcal{W}$.

[7]Similar to the actions, we support conditional XOR splits, i.e., the next task to be fired depends on the (task,action) performed at an earlier stage in the workflow. However we do not describe it here to simplify the model description.

We define a *workflow-run* as the sequence of workflow states $w_1, w_2, w_3, \cdots, w_n$ such that $w_{i+1} = \delta(w_i)$. Given a workflow W with the transition relation δ, (W_s, δ) is a transition system. From this we can derive a Kripke-structure (W_s, δ, L) by defining a labeling function. This allows us to perform model-checking on the system. The labeling function can depend on any of the information present in the workflow-state W_s. In particular, this allows us to reason about the accumulated risk as well as the tasks and actions fired in the system.

5 Tool Chain Implementation

Fig 2 shows a high level architectural diagram of the security assessment tool-chain detailing how the various components sit with respect to each other. We give a detailed description of each of the various modules in the architecture.

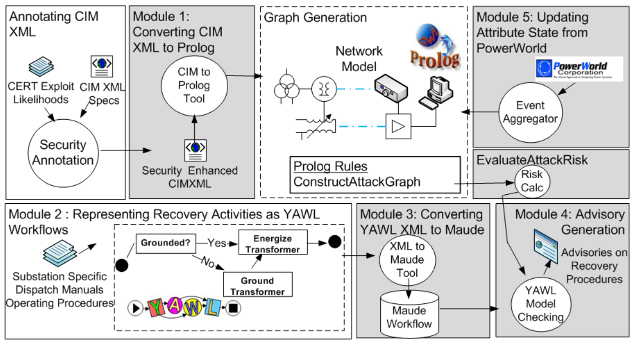


Figure 2. Tool Chain: High-Level Architectural Diagram

5.1 CIM Parsing

The network model $\mathcal{N}$ of the SCADA networks is auto-generated from annotated specifications written in the standard descriptive language based on Common Information Models (CIM) [8] with the help of a parser tool and stored in a Prolog database. CIM's comprehensive packages cover everything from equipment, topology, load data, generation profiles to measurement and scheduling. The CIM RDF schema is documented as a self-describing XML-based IEC standard. We create a mapping of the classes in the RDF model to entities in our security model. The parser identifies the main entities such as devices, connectivity and services and populates their attributes by looking at the properties and associations for each object in the CIM model. Some attributes such as privileges of users and services on the devices not covered by the basic CIM data model are manually annotated or looked up from a services to privileges table whenever a services entity is encountered. We used

```
execCode(Principal ,Victim ,Priv) :-
        device(Victim ,_,_,Svslst),
        containsVul(Svslst ,remoteExploit ,VSrvc),
        service(VSrvc,_,_,Priv ,AllowedHosts ,AllowedSvs),
        hasaccount(Principal ,Source ,PrincipalPriv),
        isIncluded(Source ,AllowedHosts),
        existsServiceType(Source,  PrincipalPriv ,  AllowedSvs),
        path(Source,  Victim ,Path).
```

Figure 3. A Prolog Rule for an Attack Graph

models of popular exploits such as buffer overflows quoted in attack graph literature and open vulnerability databases such as CERT for our vulnerabilities. For more information about the CIM to Prolog parser see [2].

Generic Prolog rules search for facts derived from the CIMs to determine whether an attack is possible. For instance the prolog rule shown in Figure 3 says that a *Principal* can execute code on a *Victim* device with a privilege *Priv* if a service *VSrvc* running on that device contains a *remoteExploit* vulnerability and it allows connections from the *Source* device that the *Principal* has an account on. Furthermore *Source* should have a service from the set of allowed *AllowedSvs* types, there should be a network path from *Source* to *Victim* and *Source* should be in the ACL of *Victim's* allowed hosts *AllowedHosts*.

5.2 Representing Recovery Workflows

We represent operating procedures of a SCADA power-grid as "generic" control-flow workflows that are not constrained by the detailed architecture of a specific power-grid implementation. For example, the workflow described in figure 4 could be applied to activate any transformer.

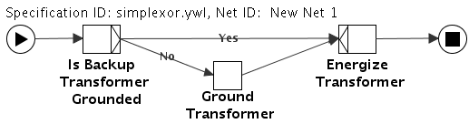

Specification ID: simplexor.ywl, Net ID: New Net 1
Yes
No
Is Backup Transformer Grounded
Ground Transformer
Energize Transformer

Figure 4. Generic Workflow Example(YAWL Editor)

The transformer workflow involves grounding the transformer (if it isn't already grounded) and energizing it afterwards. The first task of the workflow "Is Transformer Grounded" has a mutually exclusive conditional split to two other tasks. The transformer chosen for the first task (there can be multiple transformers to choose from) is left undecided. In this paper, we call the possible ways a given task can be bound to specific entities as leading to a choice of "actions". This list of actions depends on the specific substation (some substations might have 1 backup transformer and some might have more) as well as their individual states (some might already be grounded, for instance).

We describe the operating procedures as workflows using Yet Another Workflow Language (YAWL [23]). YAWL

```
 1  rl [conditionalsplit] : ——Rewrite Rule for Conditional split
 2  [<TkId, FirCond, CndActs,{[(TkId1 ? ActId1)=TkId2] ; CondSpltLst }> TkList,
 3   ActList,
 4           ( TkId1 ? ActId1 ) ActCxt ,
 5           TkIdList1 ,
 6           TkId * TkIdList2 ,
 7           TkIdList3 ,
 8           SecRsk ]
 9  =>
10  [<TkId, FirCond , CndActs,{[(TkId1 ? ActId1)=TkId2] ; CondSpltLst }> TkList,
11   ActList,
12           ( TkId1 ? ActId1 ) ActCxt ,
13           TkIdList1 ,
14           TkIdList2 ,
15           TkId2 * TkIdList3 ,
16           SecRsk ] .
```

Table 2. Maude rule describing the conditional split

is a workflow description language that supports common workflow patterns and its Editor [24] allows the construction of appropriate control-flow descriptions for the workflows and the export of this information as an XML file.

5.3 Mapping Workflows to Term-rewriting Logic

We use the "XML to Maude" converter tool to read the XML based workflow description generated by YAWL and generate a term-rewriting description of the workflow in Maude. To help with the workflow analysis, we developed a term-rewriting module in Maude. This generic module consists of rules which describe the valid workflow transitions for any generic workflow. For instance, Table 2 shows the rule that allows Maude to evaluate a task with a conditional split [8].

Lines 1-8 describe the configuration before and lines 10-16 describe the configuration after the transition rule is fired. Line 2 contains the conditional action $([(\mathsf{TkId1?ActId1}) = \mathsf{TkId2}])$ which determines whether while evaluating task TkId, the next task chosen should depend upon the action *ActId1* chosen earlier at task $\mathsf{TkId1}$. Note that $\mathsf{TkId1}, \mathsf{ActId1}$ etc., in the rules are variables and Maude can match the instantiation of the configuration to the left-hand-side of the rule and transform the current configuration as per the rewriting rule. Maude performs the transition by looking at whether the action-context (line 4) contains the tuple that says that the required condition to perform the particular task $TkId2$ was satisfied. If so, it picks the next task $\mathsf{TKId2}$ and puts it in the tasks to be fired (line 15), while simultaneously removing the current task $TkId$ from the evaluation queue (line 13). We emphasize that these rules are generic and will work for any workflow description under consideration.

The "XML to Maude" converter populates the terms to instantiate a *specific* workflow that was described generically using YAWL (and given in XML to the tool). The input includes the various actions possible at any given task and the evaluated security risk for each task.

[8] A conditional split is similar to an XOR split except that the next task chosen depends on the context

5.4 Analyzing Workflows in Term-rewriting Logic

Given the workflow description (the generic term-rewriting theory along with the specific workflow instantiation), we can use Maude's LTL Model Checker to verify any LTL property on the workflow. For instance, we can verify that any task initiated in the system should never deadlock (starting from the given initial state). This could be specified as the following LTL property:

$\Box(\mathsf{initialstate} \rightarrow \diamond(\mathsf{finalstate}))$

Given that each task might be satisfied by multiple actions, we can have multiple ways of executing a given workflow. Since each action (such as turning on a device), can have a different security risk, the risk taken to complete a given task is dependent on the choices of actions taken for any given tasks. We cannot find the workflow path (and choices taken at every task) with the minimum security risk in one run of model-checking. However, we can do that by iteratively model-checking the following LTL formula until it gives a counter example:

$\Box(\mathsf{initialstate} \rightarrow (\mathsf{FinalRisk} > \mathsf{CurrentMinRisk}))$

(where CurrentMinRisk is initialized to NoRisk and is updated until model-checking finds a counter-example)

5.5 Event Aggregator

We added support to our tool-chain to update prolog attribute information by interfacing to the PowerWorld [17] client software via COM API. A PowerWorld Client provides a graphical view of power system states made available to it by a Retriever that interfaces to real devices in a SCADA network and provides data such as bus voltages, phase angles, flows and lines and generator status. SimAuto runs on the same host as the client and provides a COM API interface to third-party applications to access the client's data-structures.

6 Evaluation

Our implementation measures approximately 1620 LOC for Prolog and 385 LOC for Maude not counting the particular network and workflow encodings. We tested it on a number of substation network-level scenarios (< 100 machines) all of which had a running time on the order of a few seconds on a regular Intel Core2Duo 2.0 GHz machine running Ubuntu Linux 7.10. We describe a scenario to illustrate our tool chain functionality to rank recovery procedures according to security risk when a fault occurs in a substation. Figure 5 shows a 275 kV substation with three transformers and two bus bars. The PowerWorld tool provides a snapshot of the system when a fault occurs on the 77kV bus-bar causing the protection system to operate, trip two of the transformers (TR2 and TR3) causing the remaining transformer to suffer from a severe overload condition. Operators in the control center have to reduce the load of the transformer in ten minutes otherwise TR1 overheats and burns out causing a blackout of the entire 77kV system in the substation.

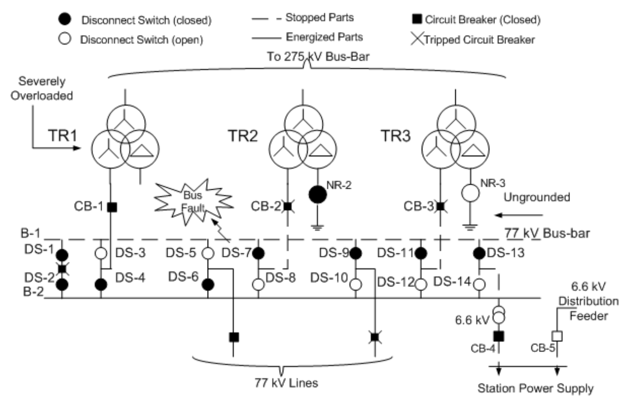


Figure 5. A 275 kV Substation with a Fault

Figure 6 shows the SCADA network that controls the energizing of the various power elements. The left hand side of the figure shows the enterprise network with commodity desktops and servers while the right hand side shows the PCN involved with taking sensor measurements and flipping switches to control the power flow. Top and bottom rows of actuators correspond to Bus 1 and Bus 2 control respectively. Groups of actuators are controlled by a PLC which is in turn controlled by a relay. Mostly serial connections (RS-485) are used to link these SCADA devices together, however some of the more upgraded devices (for instance PLC4) do use Ethernet as well. Finally the relays report their values to a data aggregator that communicates these to a data historian on the EN. The firewall contains rules to allow modbus communication between the aggregator and the historian.

We ran our security assessment toolkit on a model of this substation infrastructure. Logical attack graphs were generated for each device that was a candidate in the various recovery procedures. The attacker was assumed to be an outsider with network access to the EN and no privileges to any machine except for his own. Figure 7 shows a DoS attack graph for one of these devices: DS12. DS12 is controlled by PLC1 which is in turn controlled by Relay1 that forwards it instructions for controlling the re-energizing of the transformer. If an attacker were able to successfully deny service to this device then the underload operation for TR1 could be delayed causing it to overload.

The root node is the attack goal; in this example it is *doSattack(attacker,DS12)*, meaning “the attacker can cause

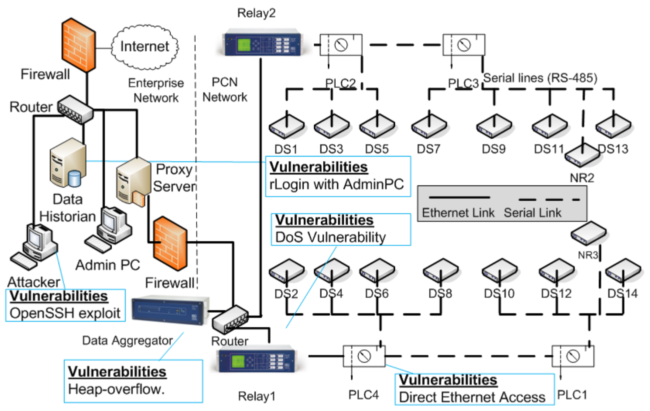


Figure 6. SCADA Network Controlling the Substation

```
<0>||--doSattack(attacker, DS12)
  <r0a> Rule: Modbus Invalid Header
   []-connectivity(aggregator,Relay1)
   []-vulExists(modbus_srvce, 'MODBUS_INVALID_HEADER', remoteExploit, doS, 0.9)
    <1>|--execCode(attacker, aggregator, root)
     <r1> Rule : Remote Heap over-flow of modbus server
     []-connectivity(historian, aggregator, modbus)
     []-vulExists(modbus_srvce, 'HEAP_OVERFLOW', remoteExploit, Integrity, 0.7)
      <2>||--execCode(attacker,historian,root)
       <r2> Rule : Trust Relationship with AdminPC
        []-connectivity(adminPC,historian, rlogin)
        []-trust(adminPC,historian)
         <3>|--execCode(attacker,adminPC, root)
          <r3> Rule : Remote Buffer Overflow
           []-connectivity(attacker,adminPC)
           []-service(adminPC,OpenSSL)
           []-vulExists(OpenSSL, 'GLSA 20071030', remoteExploit, Integrity,0.49)
   <r0b> Rule: 08 Diagnostics function code with a sub-function of 01
    []-connectivity(aggregator,Relay1)
    []-vulExists(modbus_srvce, '08DiagnosticSubf01', remoteExploit, doS, 0.87)
     |--execCode(attacker, aggregator, root) ==> <1>
```

Figure 7. A Logical Attack Graph for Device DS12

the DS12 to be unresponsive”. Every rule node is labeled with the rule name that is used for the derivation step. Rule nodes *r0a* and *r0b* illustrate that there are two possible DoS vulnerabilities (depicted by square brackets) in the Modbus protocol the Relay is using. Sending a Modbus TCP message with an invalid header or a *08Diagnostic* message would cause the Relay to be unresponsive. The aggregator that logs events, talks directly to all the relays and a root privilege on it can be achieved if the attacker can exploit a remote heap-overflow vulnerability in its modbus service. Finally the attacker can cross over the firewall to get to the aggregator by compromising the historian on the EN that has a service relationship with it. The attacker can gain access to the historian via its rlogin trust relationship with the admin PC which has an OpenSSL service with a buffer overflow vulnerability. Notice that in this scenario the attacker does not need to be an insider with user or root access to any of the machines. Not shown due to space limitation are graphs for DS8, NR2 and NR3. NR3’s risk tree is the same as that of DS12 because they are connected to the same vulnerable relay. DS8 on the other hand has a direct Ethernet access from the compromised aggregator and its integrity can be easily breached by the attacker. A high integrity risk unlike an availability risk is very dangerous as we will see in the workflows later because it means that the attacker has complete control and can thwart an operator's attempts to open or close a switch whether he does it manually or via SCADA commands.

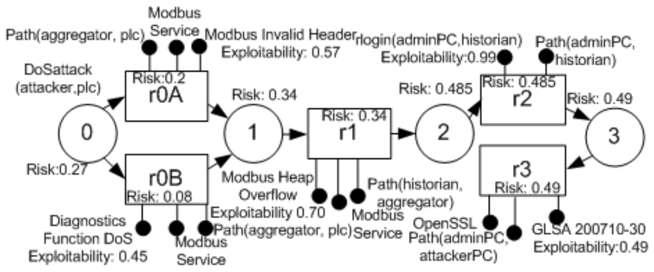


Figure 8. A Logical Attack Graph to assess the security risk of a successful DoS attack on a PLC

A more compact version of the attack graph is shown in Figure 8 that uses circles to represent derived facts, squares to represent derivation rules and filled circles to show primitive facts. This graph shows the calculation of the security label for DS12. The leaf nodes representing exploits are labeled with exploitability probabilities obtained from the CERT Vulnerability calculator [18].

Task	Context Required	Action	Risk
Select Transformer		ChooseTR2	No Risk
Select Transformer		ChooseTR3	No Risk
Ground Transformer	ChooseTR3 at Select Transformer	GroundTR3	Low Availability NR3:0.27
Close Manually	ChooseTR3 at Select Transformer	Close Switch DS12 Manually	No Risk
Close Manually	ChooseTR2 at Select Transformer	Close Switch DS8 Manually	High Integrity DS8:0.34
Close Via SCADA	ChooseTR3 at "Select Transformer"	Close Switch DS12 by SCADA	Low Availability DS12:0.27
Close Via SCADA	Choose TR2 at "Select Transformer"	Close Switch DS8 by SCADA	High Integrity DS8:0.34

Table 3. Device Risk Evaluated and Assigned to Actions

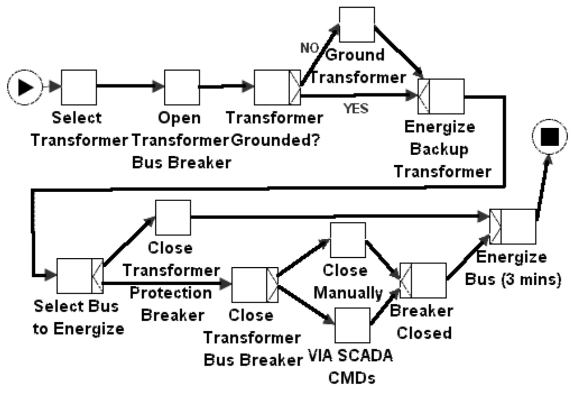


Figure 9. Procedure to Enable a Backup Transformer

Figure 9 shows the recovery procedure that an operator has to follow to bring up a transformer. These procedures are independent of the specific configuration of the sub-station and the actual running of the workflow depends on the possible actions at any given task (not depicted in the picture). For instance, the workflow task "Select Transformer" consists of two actions "ChooseTR2" and "ChooseTR3". Depending on the current configuration, the task "Transformer Grounded" can either split to "Ground Transformer" or directly to "Energize Transformer". Both splits and actions depend on the context, i.e., which transformer was chosen earlier at "Select Transformer" and whether according to the current configuration the transformer chosen was grounded or not.

A subset [9] of actions possible at any given task along with their given risks is calculated by the network-analysis tool and is given in Table 3. The list of possible actions also depends on the current state of the system. For instance, considering the configuration described in Figure 5 (TR3 is not grounded while TR2 is not), the "workflow analysis" tool can model-check if the final state is reachable from the initial state (i.e., there are no deadlocks etc.,). Furthermore, we can check if there is any path that can finish the workflow with "No Risk". Searching for the paths that fulfill the workflow in Maude gives us a list of paths each with its own risk. For instance, searching for the path with "No Risk" fails in this scenario because there is no set of choices that can fulfill the workflow with no risk at all. Searching for paths with Availability and Integrity Risks for the configuration in (Fig:5) gives us the paths as shown in in Table 6. Essentially we see that if we choose TR2, then we have a potential integrity risk [10] (because device DS8 has an integrity risk), whereas if we choose the TR3, we have a potential availability risk (because Grounding the Transformer, uses switch NR3 which has an availability risk). There are also other possible evaluations (such as using SCADA to close the switch at the task "Close Transformer Bus Breaker", which we haven't shown).

Since, the minimal risk for any workflow run is "Availability" , we recommend that the operator choose "TR3" and (due to the dependency on the transformer chosen), close switch "DS12" manually. The third workflow shown, shows the evaluation after a change in the configuration (where TR3 is grounded). Note that because of the fact that the Transformer TR3 is already grounded, we no longer need to run the task "Ground Transformer" and therefore the minimal risk run is the one that chooses "TR3" transformer (as before) and there is a set of choices which the operator can take which have no Risk at all.

[9]We do not show other actions possible at other tasks for instance "Close Transformer Protection Breaker"

[10]Recall that the evaluated risk of a given workflow run is the least upper bound of all the risks of the individual actions performed during that run

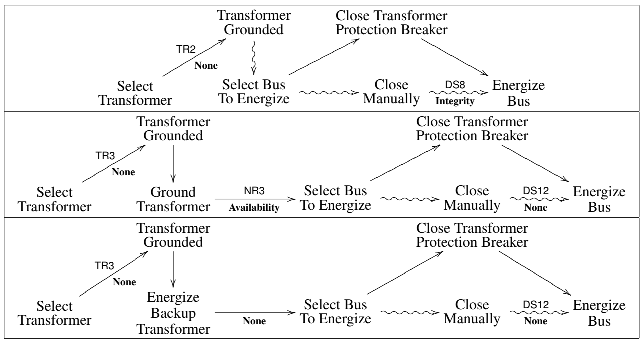


Table 4. 3 Possible ways to fulfill the workflows (Fig:9)

7 Conclusion

In this work we propose a security model that incorporates descriptions of the SCADA infrastructure and its workflow activities. By extending existing techniques for scalable attack graph generation we evaluate risks and give advisories on which workflows are safer than others based on a cost-lattice. We implemented a tool-chain that automates most of the process of generating our models from CIM specifications. Moreover the tool chain updates configuration information dynamically from an event aggregator allowing our security model to give accurate results.

In the future we would like to evaluate the scalability of our approach by using bigger models and see the impact on the model checking time. Currently actions are limited to security properties associated with devices. A more realistic model would be to incorporate side effects of actions and their impact on the state of the system. Our model could potentially allow feedback modeling (e.g. via SMTP monitoring software) of the impact of firing certain actions in a workflow and recalculation of the new security risks. The feedback mechanism would allow us to detect changes in not only status of devices but of security provisions as well. Finally we would like to expand our cost-lattice to include more properties of interest.

References

[1] British Columbia Institute of Technology Industrial Security Incident Database. *http://www.bcit.ca /appliedresearch/ security/*, 2001.

[2] Z. Anwar and R. H. Campbell. Automated Assessment of Critical Infrastructures for Compliance to Best Practices. *IFIP WG 11.10 International Conference on Critical Infrastructure Protection*, March 2008.

[3] B. Axel, R. Fredriksen, and A. Thunem. *An Approach for Model-based risk management*. Springer LNCS, 2004.

[4] E. Byres, M. Franz, and D. Miller. The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. *International Infrastructure Survivability Workshop*, 2004.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 System. In *Rewriting Techniques and Applications*, LNCS, pages 76–87. Springer-Verlag, June 2003.

[6] M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude – A High-Performance Logical Framework*. Springer LNCS Vol. 4350, 2007.

[7] U. S. N. R. Commission. Potential Vulnerability of Plant Computer Network to Worm Infection. *NRC Information Notice 2003-14*, 2003.

[8] Distributed Management Task Force, Common Information Model (CIM). *DSP 0004- CIM Infrastructure Specification 2.14*, October 2005.

[9] M. A. McQueen, W. F. Boyer, M. A. Flynn, and G. A. Beitel. Quantitative Cyber Risk Reduction Estimation Methodology for a Small SCADA Control System. *39th Annual Hawaii International Conference on System Sciences*, 2006.

[10] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

[11] J. Meserve. Sources: Staged Cyber Attack Reveals Vulnerability in Power Grid. *CNN Article*, September 2007. http://www.cnn.com/2007/US/09/26/power.at.risk/.

[12] S. Noel and S. Jajodia. Attack Graphs for Sensor Placement, Alert Prioritization, and Attack Response. *Cyberspace Research Workshop (AirForce Cyberspace Symposium)*, 2007.

[13] S. Noel, E. Robertson, and S. Jajodia. Correlating Intrusion Events and Building Attack Scenarios through Attack Graph Distances. *Computer Security Applications*, 2004.

[14] R. Ortalo, Y. Deswarte, and M. Kaaniche. Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. *IEEE Trans. Software Eng*, 25(5): 633-650, 1999.

[15] X. Ou, W. F. Boyer, and M. A. McQueen. A Scalable Approach to Attack Graph Generation. *13th ACM conference on Computer and communications security*, 2006.

[16] C. Phillips and L. Swiler. A Graph-based System for Network-Vulnerability Analysis. *New Security Paradigms Workshop*, 1998.

[17] POWER WORLD The Visual Approach to Analyzing Power Systems. *http://www.powerworld.com/*, July 2007.

[18] M. Schiffman, G. Eschelbeck, D. Ahmad, A. Wright, and S. Romanosky. CVSS: A Common Vulnerability Scoring System. *http://www.first.org/cvss/cvss-guide.html*, 2007.

[19] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated Generation and Analysis of Attack Graphs. *IEEE Symposium on Security and Privacy*, 2002.

[20] Stephanou and Tony. Assessing and exploiting the internal security of an organization. *The SANS Institute*, 2001.

[21] U. S. C. E. R. Team. Us-cert vulnerability note field descriptions. *http://www.kb.cert.org/vuls/html/fieldhelp*, 2007.

[22] C. W. Ten, C. C. Liu, and M. Govindarasu. Vulnerability Assessment of Cybersecurity for SCADA Systems Using Attack Trees. *Power Engineering Society General Meeting*, July 2007.

[23] W. M. P. van der Aalst and A. H. M. ter Hofstede. Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275, 2005.

[24] QUT BPM Research group and Eindhoven University's YAWL Editor. *http://www.yawl-system.com/*, 2007.