World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:6, 2010

Access Policy Specification for SCADA Networks

Rodrigo Chandía and Mauricio Papa
Institute for Information Security
Computer Science Department, University of Tulsa
800 S. Tucker Dr., Tulsa, OK 74104

Abstract—Efforts to secure supervisory control and data acquisition (SCADA) systems must be supported under the guidance of sound security policies and mechanisms to enforce them. Critical elements of the policy must be systematically translated into a format that can be used by policy enforcement components. Ideally, the goal is to ensure that the enforced policy is a close reflection of the specified policy. However, security controls commonly used to enforce policies in the IT environment were not designed to satisfy the specific needs of the SCADA environment. This paper presents a language, based on the well-known XACML framework, for the expression of authorization policies for SCADA systems.

Keywords—Access policy specification, process control systems, network security.

I. Introduction

Supervisory Control and Data Acquisition (SCADA) systems allow human operators to remotely observe the state of a physical process and command actuators to effect change on the process. Early SCADA systems, in most cases, consisted of a mainframe computer, connected to a number of simple remote devices. These devices had the ability to access sensors and send commands to actuators located on the physical process of interest. Modern SCADA systems comprise networks of servers – providing opportunities for complex interaction with expert personnel – connected to a distributed network of remote intelligent devices. These intelligent devices control sophisticated digital sensors and actuators.

A SCADA control network transports and delivers messages from the central control facility to the remote devices and back. A number of protocols exist for use in SCADA networks. Common protocols include DNP3 [28], Modbus [19] and IEC 61850 [10]. Most of these protocols originally relied on serial links for transport. Vendors and standards organizations have recently started to support Ethernet and TCP/IP as the underlying transport layer. Modbus/TCP [18] and DNP3 over TCP [29] are examples of standard protocols updated for this purpose.

SCADA protocol design focused on operational functionality and relied on physical isolation for security. However, the use of TCP/IP as a transport mechanism for control messages may render a process control system vulnerable to cyber attacks or malicious manipulation. For that reason, the development of cyber security tools capable of mitigating the risks (associated with both internal and external threats) has been recognized as an important concern for any enterprise SCADA system.

Most SCADA security standards deal with risk mitigation and technical security controls at the central management facility. This is in part due to the alarming trend of enterprises connecting, perhaps indirectly, SCADA systems to the corporate network using TCP/IP. Securing the SCADA control network at the field level has received comparatively less attention. Anyone tapping to an unsecured field network would be able to disrupt sections of the process under control. Large scale SCADA systems frequently span large distances, making it very difficult to secure the field network by employing physical means. Even for smaller SCADA installations complete physical security is hard to accomplish. This makes cyber security a critical component of the overall security posture of SCADA systems.

Policy enforcement efforts at the central control facility use procedures and controls similar to those available to a typical IT network. However, policy enforcement at the SCADA control network poses unique challenges not typically faced in the IT world: (i) prevalent use of serial lines and legacy protocols prevent the use of firewalls and network IDS systems; and (ii) SCADA protocols, by design, operate under the assumption of a trustworthy environment. This complicates the implementation of authorization and authentication for users and devices. Operators often lack the tools necessary to enforce security policies and may be opposed to the imposition of security controls that may negatively affect performance, reliability and availability. These factors present a difficult obstacle to overcome for the mitigation of security risks on these networks.

The approach described in this paper addresses this concern by proposing a framework that enables authorization policy auditing and enforcement. A key element of the framework is a policy description language specifically designed to protect SCADA devices and networks. The main goal of this effort and the proposed policy language is to provide a simplified path from policies to enforcement, thus improving the overall security posture of the SCADA control network.

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.

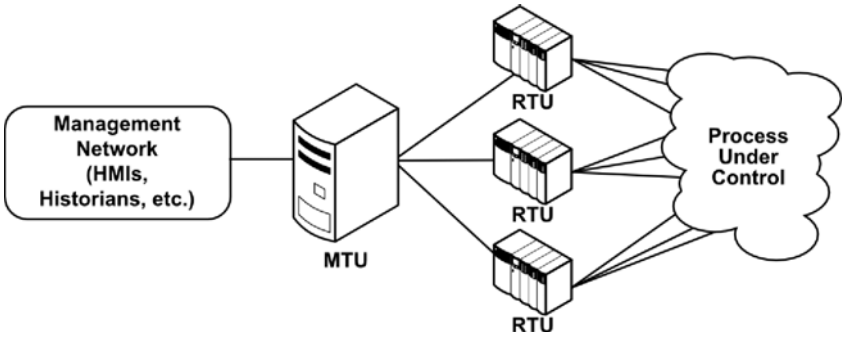

Fig. 1: Prototypical SCADA control network.

II. SCADA Networks

SCADA systems typically consist of (i) a central facility which hosts devices that presents readouts and schematic information about the process state, along with controls to modify process actuators (also referred as the Management Network); (ii) a *SCADA control network* providing message transport services and containing control devices as network endpoints; and (iii) the process machinery, with sensors and actuators operated by these control devices. Messages transported by the SCADA control network primarily convey state information of the process under control and, when required, they may also convey instructions to modify actuator settings that effect changes on the process.

Boyer [5] describes SCADA control networks as composed by a Master Terminal Unit (MTU) connected to a number of Remote Terminal Units (RTUs). The MTU generates reading requests to update the information presented to the operators and write requests to modify actuators upon operator request. RTUs provide sensor readings from its inputs (reads) and modifies actuator settings (writes) on its outputs upon request from the MTU. Figure 1 depicts a simplified SCADA control network.

The architecture assumes the MTU frequently communicates with Human Machine Interface (HMI) devices and other supporting devices on the management network. One or more operators – expert personnel trained to manage the process under control – use the HMIs to determine the process state and command process actuators. This is abstracted as a single box at the left side of Figure 1.

Many common SCADA protocols use a request-response message exchange pattern to control the transmission of messages between the MTU and RTUs. Request-response protocols force an RTU to only generate a message as a direct response to a request. This provides a simple way to deal with congestion control issues and to recuperate from transmission errors. Typical examples of these protocols include DNP3 [28], Modbus [20], [17] and proprietary protocols implemented by each vendor such as ROC [9] and BSAP [8]. Note that these protocols sometimes provide alternative exchange patterns. For example, devices using the DNP3 protocol may spontaneously generate messages to convey alarm conditions, and Modbus requires devices to not reply to broadcast messages; but these are exceptional conditions rather than normal communication modes. Other types of process control such as Distributed Control Systems (DCS) use protocols for process control based on other exchange patterns. Controller Area Networks (CAN) [7], for example, use publish-subscribe patterns to control message distribution.

Modern SCADA systems operate within the environment of an enterprise. Communication occurs via paper-based procedures or system-to-system communication. Figure 2 presents a high level representation of a modern SCADA control network within an enterprise (loosely based on the ISA-95 standard [13]). A corporate network contains enterprise management systems used by billing, sales and procurement; and systems for production management and process optimization. Corporate systems require selected sensor readings for billing, procurement and performance analysis purposes, and provide high-level guidance for the operation of the process equipment.

Current security standards and SCADA security practitioners consider connectivity to the the corporate network a security risk which needs to be addressed. A DMZ (a De-Militarized Zone) implements a common measure for the mitigation of these risks [6]. The DMZ is a subnetwork located between trusted and untrusted networks. The DMZ is separated from the other networks by firewalls. The DMZ contains only hosts which require connectivity to both trusted and untrusted networks. The firewalls limit the traffic to the minimum necessary, and prevent any traffic to flow between the trusted and untrusted network. This serves to isolate the trusted network from dangerous traffic and allow administrators to focus security measures on the hosts located within the DMZ.

The SCADA system comprises the management network and the control network. The management network hosts the process management services. These include HMIs, historians, patch management systems and protocol servers acting as MTUs. These are the main systems used by operators to determine current process status, alarm conditions and modify actuator settings. One or more MTUs interconnect these systems to the control network.

The control network provides communication to the RTUs. This network, mostly designed to convey messages between the MTUs and the RTUs, also provides protocol translation facilities. RTUs directly interface with sensors and actuators on the process machinery.

Gateways address the problem of converting messages between different protocols. Situations where two parts of the network use different protocols abound. Protocols used in

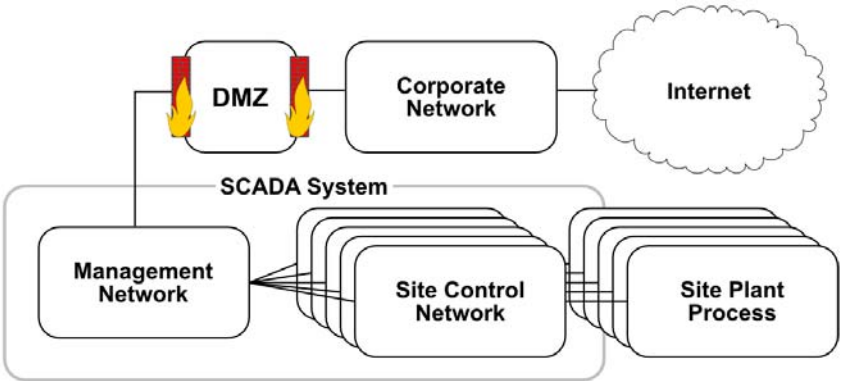

Fig. 2: Schematic representation of a typical SCADA control network.

the management network typically differ from those used in the control network; a merger between two enterprises may require an existing management facility to connect to a new network; vendors may obsolete equipment over time, making different devices (using a different protocol) a less expensive proposition than acquiring the current models; or even by design, a chosen vendor for a type of devices (intelligent measurement sensors) might require a protocol not appropriate for the communication links used to communicate with the MTU.

TCP/IP [24] has become a common protocol for carrying messages over long and short distances. Its ease of use and low cost make it an attractive alternative. SCADA protocols need special consideration when using TCP/IP as its underlying transport.

Encapsulation provides a simple solution for the transport of serial messages over TCP. A TCP connection to a terminal server, at a predetermined port, replaces the serial link; TCP messages encapsulate the serial messages, allowing transport over an IP network. The terminal server receiving the connection acts as a gateway by converting messages between the TCP stream and a serial link.

Vendors and standards organizations also provide mechanisms and specifications for direct transport of SCADA messages over TCP/IP, e.g., Modbus/TCP standard [18].

A. SCADA Network Reference Model

To design the proposed policy enforcement architecture, a reference model for a SCADA network was needed. This network contains a management network, a control network and a physical process under control. The proposed architecture concentrates on policy enforcement for the control network. It does not contemplate a detailed architecture for the management network beyond what is presented in this section.

The SCADA network contains one MTU connected to one gateway. This gateway provides protocol translation and routing as needed to interconnect the MTU to one or more SCADA sites. A SCADA site contains a set of RTUs interconnected by a single network. In most cases a multidrop serial link provides connectivity between the gateway and the RTUs. Figure 3 shows a depiction of the SCADA control network model.

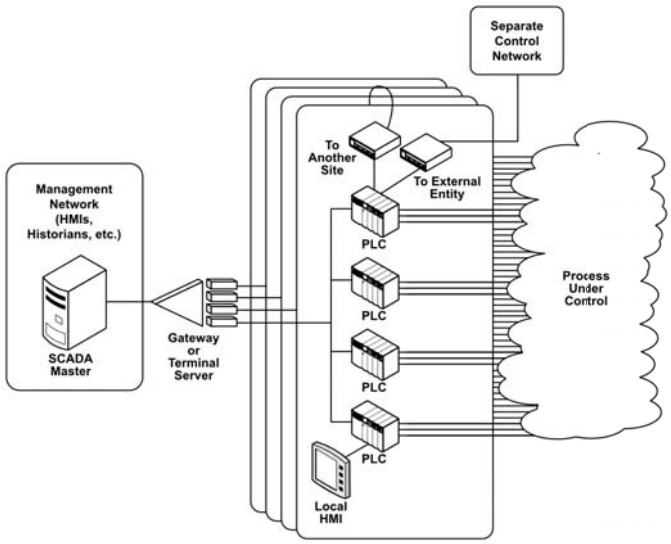

Fig. 3: A Model for a typical SCADA control network.

The model contemplates a hierarchical organization for message distribution. All messages start at the MTU, which sends the message to the gateway. The gateway then routes and, if necessary, translates the message to the appropriate site network. All RTUs in the SCADA site might observe the message, but only the one matching the destination takes action. If the destination field in the message specifies a broadcast address, all RTUs accept the message and take action. After performing the requested action the receiving RTU responds to the message; unless the message was destined to a broadcast address and the protocol prohibits responses for such messages.

The model network also contemplates a number of exceptions to this hierarchical message distribution. Some RTUs directly send and receive messages to other entities. The other party in this case may either be a RTU in the same site, a local HMI on the same site, another entity on a different site, or even an entity on a separate control network. These exceptions model common situations such as operators and workers that need local supervision and control of process

machinery; RTUs that require direct communication to another RTU without intervention from the management network; and vendors and contractors which require remote access for maintenance and troubleshooting.

Without loss of generality this SCADA model leaves out the situation where a MTU connects to several gateways. For the purposes of this research, these are modeled as separate SCADA control networks where each network contains a separate MTU.

SCADA security standards provide guidance on managing security in the management network. The ISA organization has published documents to help practitioners secure process control networks networks. These documents provide a model for the segregation of systems into security zones [14] and a collection of technologies to help operators select appropriate security controls [15]. More recently, NIST has published a draft document [26] that collects industry wide practices to guide the mitigation of security risk. Options for the mitigation of security threats on the SCADA control network are still limited. The AGA-12 standards [1], [2] require the use of encryption for the control network and define a specific protocol for the protection of serial links. As wireless technologies make inroads into SCADA control networks, more operators require the use of encryption over 802.11 links. Finally, some standard protocols including DNP3 [29] and IEC 62351-5 [11] have been extended to include secure authentication and encryption in the protocol. Security policy enforcement at the field level remains an open problem.

III. Policy Models and Authorization Policies

Information security policies provide high-level plans to satisfy goals for the protection of the information handled by an enterprise [3]. As such, these policies reduce risk related to the information handled by the enterprise by describing goals of procedures, determining assets to protect, placing limits on controls to implement, guiding product selection, delineating best practices to follow during software development, and other practices and controls.

Enterprises enforce the implementation of policies using procedures and technical controls. Technical controls provide automated mechanisms to implement the objectives stated in the policies. These controls range from door locks to event correlation software.

Security policies also guide the development of practices, procedures and controls to minimize risk on SCADA systems. The main concern for these systems lies not in data loss, but on loss of control. As a result, most security and safety measures relate to the appropriate handling of process events, abnormal conditions, safety procedures that minimize downtime, production losses, and safety of personnel and the general public. Security from malicious activity has traditionally been a secondary concern in the industrial environment. In particular, operators avoid technical and management controls that may impede the operation of SCADA devices or prevent personnel from performing its duties.

Authorization policies, a specific part of security policies, form a central part of this research. XACML [22] provides an example of a generic language for the expression of authorization policies that can also be used to mechanize policy evaluation. Section IV describes this language that provides the inspiration for our design of a policy language targeting SCADA systems. Existing implementations [27] for policy evaluation are readily available.

Authorization policies provide rules to determine whether a subject can perform an action on a specific resource, given the current environment. For example, a SCADA policy may allow only a specific group of HMIs and MTUs (subjects) to start and stop (actions) an engine (the resource) but only during a certain part of the day (environmental condition).

Several solutions exist that attempt to centralize and ease the management of enterprise-wide authorization policies. Directory services such as NIS [25] and LDAP [32] allow enterprise-wide management and distribution of authorization information for authorization decisions. However, each system still makes its own authorization decisions based on its own implementation of authorization policies. Such policies might not align with the actual policies defined by the enterprise. For example, a policy may limit plant workers from accessing an HMI when not in their shifts, but the HMI may not take this parameter into account when restricting operator login. Mechanisms to express flexible authorization policies throughout the enterprise are only now appearing, with XACML [22] being one of the most domain independent (see Section IV).

SCADA systems usually follow a policy that allows open access for any personnel present near the equipment. The implicit assumption is that only authorized personnel are allowed into the central control facility and near other sensitive - and frequently dangerous - areas of a plant; access in this case is controlled by physical means. Moreover, having operators spend time supplying credentials to access an HMI is deemed a safety risk. Access control for messages between devices has traditionally been deemed unenforceable, even though SCADA traffic is generally well defined, repetitive and unchanging under normal operations. Interrupting traffic may adversely affect the reliable operation of the process.

IV. Extensible Access Control Markup Language - XACML

XACML (eXtensible Access Control Markup Language) facilitates the expression of interoperable authorization policies [22]. This language allows the definition of specific access control policies for diverse resources, subjects and actions in a common language irrespective of the actual implementation. An access request is granted or denied by combining enterprise-wide policies with attributes of the subject, action, resource and environment involved in the access attempt. Enterprises using XACML define authorization policies based on their own requirements rather than being forced to follow what specific systems implement. Work into the definition of XACML started during April 2001 [4] by the OASIS Foundation [23] under Technical Committee XACML. OASIS published the first version of the document in February 2003 [21]. The current version is 2.0 [22], published in February 2005.

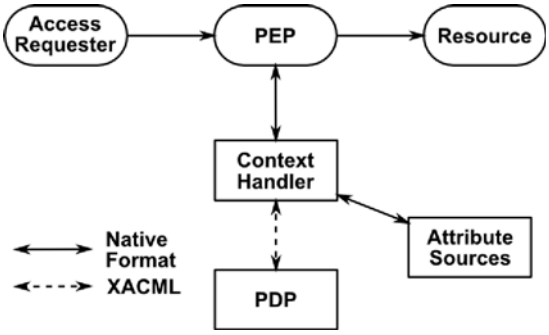

Fig. 4: XACML architecture.

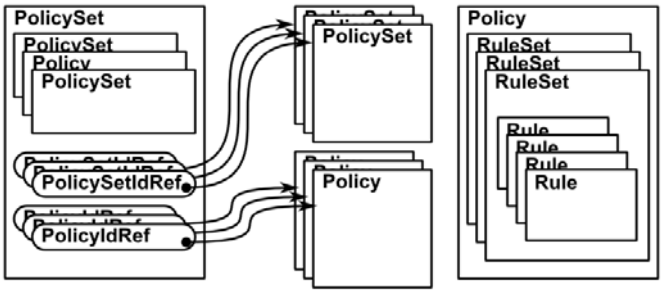

Fig. 6: Relationship of XACML top level elements.

TABLE I: Entities in the XACML architecture

Policy Decision Point (PDP)	Evaluates a XACML policy and produces an authorization decision.
Context Handler	Acts as an intermediary between the PDP and all external resources. Converts between each entity native format and XACML with the PDP. [31]
Policy Enforcement Point (PEP)	Stands between an access requester and a resource. Produces decision requests and enforces authorization decisions. [31]
Policy Information Point (PIP)	Provides attribute values to be used by the PDP for policy decisions.
Access Requester	Some entity requesting access to a resource for the execution of a specific action.
Resource	A service, system component or data.
Attribute Sources	A set of entities that provide attributes on demand for the environment, resources, actions or subjects.

A. Authorization Model

The XACML standard uses a simple architecture (summarized in Figure 4) as a basis for the definition of roles involved in the processing of XACML documents. This architecture is not mandatory, it serves as a representative environment where XACML operates.

Under this architecture, a Policy Enforcement Point (PEP) [31] stands between a resource and an access requester (see Table I). The PEP validates any credential and attributes provided by the access requester, but defers any access decision to the PDP.

As the native format for decision requests may not be XACML, all such requests pass through the Context Handler before reaching the PDP as a XACML decision request. Figure 5 shows the sequence of messages generated by the architecture elements from the moment when the PDP sends a decision request.

From the decision request the PDP determines the set of policies necessary to provide a response. This in turn determines action, subject, resource and environmental attributes needed for the resolution of the predicates contained in the relevant set of policies that apply to the decision. The decision request contains some of the attributes, the PDP queries the Context Handler for a resolution of the remaining ones. The Context Handler attempts the resolution of said attributes by forwarding attribute queries to one or more Policy Information Points (PIPs). Optionally, the resource also sends the resource content to the Context Handler to augment the resource attributes available for the decision. The Context Handler then sends a response with all the attributes it gathered to the PDP. Finally, the Context Handler forwards the response to the PDP in its native format. The decision response may optionally contain a clause stating obligations the PDP must carry before granting access. If the PDP cannot fulfill the obligations clause access should be denied.

B. Policy Structure

XACML defines several elements that form the overall authorization policy (Figure 6). A *policy set* defines a set of *policies* to evaluate for a policy decision. A *policy set* may contain a number of *policies* or several *policy sets* either directly or by reference. A *policy*, contains one or more *rule sets*, which in turn contain one or more *rules*. *Policy sets* also define which of the available algorithms to use for the combination of individual policy decisions. Similarly, *rule sets* define which of the available algorithms to use for the combination of individual rule evaluation results.

Rules are expressions that return a value when evaluated. When a *rule* cannot be evaluated due to attributes mentioned with undefined values, it returns an *indeterminate* result. Available elements in a rule include boolean operators, relational operators, arithmetic operators, string handling functions, set operators, XPath selection, conditional evaluation, references to other rules and attribute values. For a *rule* to provide a decision, its *rule set* must evaluate to a boolean result of *true*. Otherwise the result is *not applicable*.

Each *policy set* and *policy* may contain a clause expressing the conditions under which the policy set or policy apply. This expression, allows a PDP to quickly determine which policies do not apply for the decision and avoid evaluating the *policy sets*, *policies* and *rule sets* contained within.

PEPs are responsible to interpret policy decisions, allow or prevent access, and take appropriate actions. *Permit* and *deny* decisions have obvious meanings, but each PEPs may interpret *not applicable* and *indeterminate* decisions in different ways.

XACML provides a flexible way to organize policy statements, a rich set of operators and a mechanism to identify

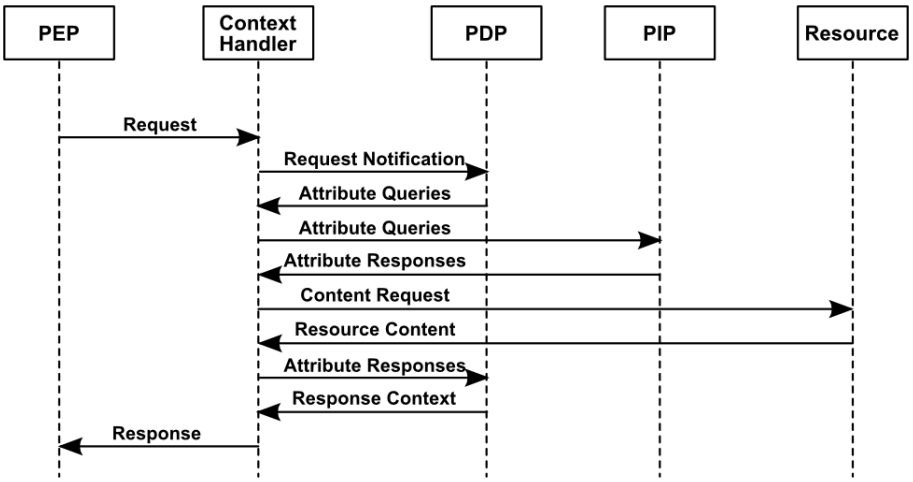

Fig. 5: Messages exchanged by elements of XACML model.

values from external repositories. This language allows the expression of versatile policies abstracted from implementation details. SCADA devices, having a wide variety of competing implementations can benefit from using this language as an unifying option for the expression and enforcement of authorization policies.

On the other hand, XACML as a descendant of XML is primarily a machine oriented language. As such, direct human interaction is not one of its features. Directly maintaining a XACML policy is not a reasonable expectation. XACML's many features also add complexity. Users of XACML may not need all its features. For this reason, our approach uses its own policy language (loosely modeled after XACML). A compiler from the policy language to XACML allows the use of a subset of XACML features to perform the policy evaluation functionality.

V. Policy Language

This section describes a policy language created for the definition of *policy statements* that conform to the overall policy of the enterprise. A Policy Decision Point (PDP) evaluates each of these statements upon the reception of an authorization request from a Policy Enforcement Point (PEP), as shown in Figure 7. The results of these evaluations are combined to produce an overall authorization decision which is returned to the PEP. Policy statements reference a set of attributes. To evaluate policy statements, the PDP obtains actual values associated with these attributes from the authorization request and from a set of attribute repositories.

This language defines two kinds of policy statements; those that produce a "permit" decision and those that produce a "deny" decision. The evaluation of a policy statement produces one of three decisions: its stated decision ("permit" or "deny"), "not applicable" or "indeterminate." A result of "not applicable" indicates that the policy statement did not produce its stated decision. A result of "indeterminate" indicates that a problem occurred during the evaluation that prevented the determination of the policy statement applicability. The expected behavior of the evaluation is to either determine that the policy is applicable (producing its stated result) or "not applicable." Indeterminate results provide information on the cause of the problem and help debug errors in the definition of policy statements and problems with the definition of stored attributes. The applicability of a policy statement is determined by the computation of boolean expressions contained inside.

The PDP determines the overall decision by combining the individual results from each statement using a pre-defined heuristic. A suitable heuristic is presented in Section V-B.

In addition to the authorization result, policy statements deemed applicable also contribute *obligations* to the overall decision result. These obligations prescribe actions the PEP must execute before granting or denying authorization. Obligations are associated to the "permit" or "deny" value declared within its containing policy statement. The PDP discards obligations that do not match the overall "permit" or "deny" result.

The following two subsections describe relevant parts of the policy language and the heuristic to use for the combination of evaluation results.

A. Policy Statement

A policy statement contains up to four sections. (i) The attributes section declares the attributes used within the policy statement. (ii) The precondition specifies combinations of attribute values necessary to proceed with the evaluation of the remaining sections of the policy statement; the policy statement is deemed "not applicable" otherwise. (iii) The condition section specifies the result to return ("permit" or "deny") and a boolean expression that determines the policy statement applicability (true produces the stated result and false produces "not applicable"); this is the only mandatory section. (iv) Finally, the obligation section lists instructions for the PEP to execute when the policy statement is deemed applicable and the result of its evaluation matches the overall authorization decision. Problems in the evaluation of (ii) or (iii) produce a result of "indeterminate."

The following rule expands the start symbol of the grammar into the symbols which in turn expand to each of the policy

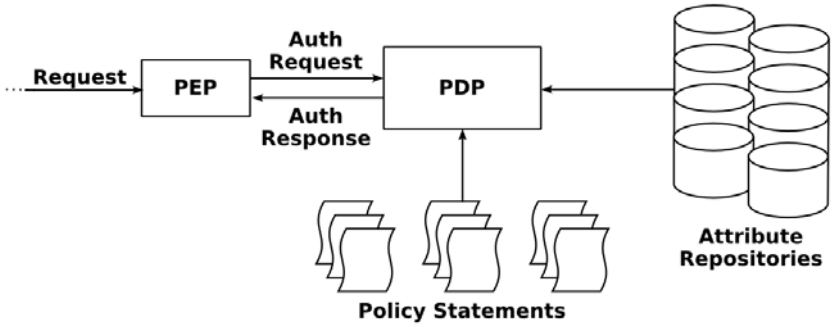

Fig. 7: Participants of the policy evaluation process.

```
/* Attributes section */
using
    subject     uri subject_id
    action      uri action_id
                uri parameter
    resource    uri resource_id
    environment time timestamp
/* Precondition section */
when
    action   action_id = uri("modify")
             parameter = uri("temperature")
             or
             action_id = uri("modify")
             parameter = uri("pressure")
    resource resource_id = uri("boiler")
/* Condition section */
deny if
    not subject_id = uri("master") and
    timestamp > time("16:00") and
    timestamp < time("17:00")
/* Obligation section */
then
    log("% attempted to access the
        boiler temperature", subject_id)
```

Fig. 8: An example of a policy statement.

sections:

$$policy \rightarrow symbols?\ scoping?\ action\ consequence?\ \mathbf{EOF}, \quad (1)$$

where $symbols$ expand to the attributes section, $scoping$ expands to the precondition section, $action$ expands to the condition section and $consequence$ expands to the obligation section.

The grammar allows the expression of parts of an authorization policy such as "only the master station can modify the boiler temperature and pressure between 4:00 PM and 5:00 PM." The policy statement in Figure 8 would allow such operations, assuming the PEP and the attribute repositories provide the referenced attributes. Here we also assume that other policy statements allow the master to modify the boiler temperature and pressure regardless of the time.

The following sections define a machine readable language to express the $symbols$, $scoping$, $action$, and $consequence$ sections of a policy statement.

Attributes Section A non-trivial policy statement makes reference to attributes associated to subject, action, resource or environment. This section of a policy statement provides a compact notation for the attributes referenced in a policy statement, independent of the underlying implementation.

$$symbols \rightarrow \text{“}\mathbf{using}\text{”}\ (\ var\_def\ )^{*} \quad (2)$$

$$var\_def \rightarrow \text{“}\mathbf{subject}\text{”}\ data\_type\ \mathbf{SYM}\ subject\_xacml\_spec? \quad (3)$$

$$\rightarrow (\ \text{“}\mathbf{action}\text{”} \mid \text{“}\mathbf{resource}\text{”} \mid \text{“}\mathbf{environment}\text{”}\ )\ data\_type\ \mathbf{SYM}\ xacml\_spec? \quad (4)$$

Attributes are written as a white space separated list of attribute declarations after the keyword "**using**" (rule 2). Each attribute declaration stemming from the grammar symbol var_def defines whether the attribute belongs to subject, action, resource or environment, its data type, its name, and optionally implementation dependent parameters. These implementation dependent parameters are represented by the symbols $xacml\_spec$ and $subject\_xacml\_spec$ in rules 3 and 4. They associate the attribute with the naming parameters used by the underlying implementation. For example, the attribute `subject_id` in Figure 8 could require an implementation dependent specification like:

```
/* Attributes section */
using
    subject uri subject_id
       = ("urn:scada-policy:subject-identifier")
    ...
```

Here a subject attribute of type URI named `subject_id` is declared. This declaration explicitly instructs the PDP to use the name `urn:scada-policy:subject-identifier` when referring to the `subject_id`.

Precondition Section Authorization policies may contain numerous individual policy statements. One of the main issues for a PDP is to quickly reduce the number of statements to evaluate. This section allows the expression of conditions on attribute values to determine whether the policy statement evaluation should proceed to the next section, where more complex expressions may occur. This section allows a Policy Decision Point (PDP) to implement efficient indexing and

retrieval of statements which may apply to a decision.

$$scoping \rightarrow \text{“}\mathbf{when}\text{”}\ (\ scope\_element\ )^* \quad (5)$$
$$scope\_element \rightarrow \text{“}\mathbf{subject}\text{”}\ scope\_disj \quad (6)$$
$$\rightarrow \text{“}\mathbf{action}\text{”}\ scope\_disj \quad (7)$$
$$\rightarrow \text{“}\mathbf{resource}\text{”}\ scope\_disj \quad (8)$$
$$\rightarrow \text{“}\mathbf{environment}\text{”}\ scope\_disj \quad (9)$$
$$scope\_disj \rightarrow scope\_conj\ (\ \text{“}\mathbf{or}\text{”}\ scope\_conj\ )^* \quad (10)$$
$$scope\_conj \rightarrow scope\_oper+ \quad (11)$$
$$scope\_oper \rightarrow symbol\ (\ \text{“=”}\ |\ \text{“<”}|\ \text{“>”}|\ \text{“<=”}| \quad (12)$$
$$\text{“>=”}\ )\ constant \quad (13)$$
$$\rightarrow \text{“}\mathbf{regexp}\text{”}\ \text{“(”}\ symbol\ \text{“,”}\ constant\ \text{“)”} \quad (14)$$

Rules 6 to 9 group attribute specifications by attribute type (subject, action, resource and environment). Semantic checks (not shown) ensure that subject, action, resource and environment groups appear at most once. These in turn contain a disjunction of clauses separated by the “**or**” keyword. Each clause contains a list of relations between attributes (on the left hand side) and constants (on the right hand side). These lists form logical conjunctions where all attribute values must match the related constants. For multi-valued attributes (see Section V-A), the operations match if any value matches the constant.

Policy statement evaluation continues to the condition section only if this section evaluates to *true*; otherwise the evaluation produces a result of *not applicable*.

For example, this precondition (from Figure 8)

```
/* Precondition section */
when
    action   action_id = uri("modify")
             parameter = uri("temperature")
             or
             action_id = uri("modify")
             parameter = uri("pressure")
    resource resource_id = uri("boiler")
    ...
```

allows to proceed with the evaluation of the policy statement only if `resource_id`, and `action_id` match the specified values, and the `parameter` being modified is either the `pressure` or the `temperature`.

This is equivalent to the expression:

$resource\_id = uri(\text{“}boiler\text{”}) \wedge ((action\_id = uri(\text{“}modify\text{”}) \wedge parameter = uri(\text{“}temperature\text{”})) \vee (action\_id = uri(\text{“}modify\text{”}) \wedge parameter = uri(\text{“}pressure\text{”})))$

Condition Section This section describes core components of the policy statement. It defines the type of decision (“permit” or “deny”) the policy statement produces (rule 15). It also defines under which combination of attributes and constants the policy statement is applicable (based on the *expression* in rule 16).

$$action \rightarrow (\ \text{“}\mathbf{permit}\text{”}\ |\ \text{“}\mathbf{deny}\text{”}\ )\ condition \quad (15)$$
$$condition \rightarrow \text{“}\mathbf{if}\text{”}\ expression \quad (16)$$

The $expression$ within this section must return a boolean data type. A result of *true* causes the policy statement to evaluate to the action defined in rule 15. A result of *false* causes the policy statement to evaluate to *not applicable*.

The following condition section (taken from Figure 8) declares that evaluating the policy statement as applicable produces a “deny” decision. This happens only if the previous precondition is true and the $subject\_id$ is equal to “master” and the request timestamp occurs between 4:00 PM and 5:00 PM.

```
/* Condition section */
deny if
    not subject_id = uri("master") and
    timestamp > time("16:00") and
    timestamp < time("17:00")
```

Obligations This optional section mandates actions the PEP must execute after the policy statement is deemed applicable. These obligations will be executed by the PEP only if the result of “permit” or “deny” from this policy statement matches the overall result (obtained after evaluating and combining the results of all the policy statements). For example, the “**log**” obligation in Figure 8 will be executed by the PEP only if the policy statement is applicable and the result of combining the evaluation of all policy statements also produces a “deny.”

$$consq \rightarrow \text{“}\mathbf{then}\text{”}\ (\ consq\_expr\ )^* \quad (17)$$
$$consq\_expr \rightarrow \text{“}\mathbf{log}\text{”}\ \text{“(”}\ \mathbf{STRING}\ (\ \text{“,”}\ expression\ )^*\ \text{“)”} \quad (18)$$
$$\rightarrow \text{“}\mathbf{forward}\text{”} \quad (19)$$
$$\rightarrow \text{“}\mathbf{store}\text{”}\ \text{“(”}\ bag\_symbol\ \text{“,”}\ expression\ \text{“)”} \quad (20)$$
$$\rightarrow \text{“}\mathbf{exec}\text{”}\ \text{“(”}\ (\ bag\_symbol\ \text{“,”}\ )^*\ \mathbf{STRING}\ \text{“)”} \quad (21)$$

The “**log**” obligation in rule 18 causes the Policy Decision Point (PEP) to register a message in the log using a specific message and values. These values result from the evaluation of expressions based on the same context used for the rest of the policy statement. Occurrences of the wild-card “%” are substituted with the result of the evaluation. Occurrences in excess of the number of expressions are substituted with the string “`(undefined)`”. Expressions in excess of the number of wildcards are ignored.

The “**forward**” obligation in rule 19 causes the PEP to forward the request to a higher level PEP for further processing. The mechanism for the transmission and the effect this has on the other PEP is up to the implementation.

The “**store**” obligation in rule 20 causes the PEP to modify an attribute repository by storing the result of an expression into a specific attribute. This allows the encoding of state into the policy statements.

The “**exec**” obligation in rule 21 allows the definition of other consequences as a script to execute under some predefined scripting language. The script has access to all the attributes used to evaluate the policy statement.

Expressions Expressions provide the ability to compose attribute values using predefined functions. The main use of functions is to generate a boolean value to determine the applicability of a policy statement. Expressions are also used to provide the values used in the obligations section of the policy statement. The policy language defines the usual boolean, relational and arithmetic operators, shown in Table II. The compiler automatically casts integers into doubles as needed.

The language recognizes constants of type integer, double, boolean and string. For all other data types (uri, date,

TABLE II: Policy language operators and precedence

Name	Notes
or	
and	
"**not**"	
<, >, <=, >=, =	All data types may be tested for equality
+	
-	
*, /, "**mod**"	"**mod**" only accepts *integer* parameters
∼, function	"∼" provides set equality
- (negative)	
( )	

time, dateTime, dayTimeDuration, yearMonthDuration, ipAddress and dnsName) the language provides a function of the same name as the data type. These functions take a string as a parameter and returns the corresponding data type. An URI constant, for example, may be expressed as `uri("http://example.com")`.

Multi–valued Attributes Authorization requests and repositories may provide multiple values for a single attribute. This is represented in the language with the concept of a *bag*. Bags are unordered collections of values with possibly non-unique entries of the same data type.[1]

Multi-valued attributes provide an useful abstraction. For example, a subject may hold the roles of "master" and "slave" at the same time; or a resource may allow both "read" and "write" actions.

Several functions return or take bags as parameters. The *one* function and the *bag* function, for example, allow the conversion between a multi-valued and a single-valued environment.

The language regards attributes and expressions as single valued or as bags depending on the context under which the attribute is referenced. The compiler allows the conversion between bags and single values at run-time, with the caveat that bags in single valued contexts must hold only one value. An error occurs otherwise, producing *indeterminate* as the statement evaluation result.

The following statement shows an expression where the an attribute $addresses$ is referenced first as a bag and then as a single value:

```
size(addresses) = 1 and addresses = 5000
```

The $size$ function takes a bag and returns the number of elements it contains. This protects the evaluation of `addresses = 5000`. The equality operator takes single values on both sides; an "indeterminate" result would occur if $addresses$ contained multiple values when evaluating the equality.

B. Policy Statement Combination

The PDP combines the results of the evaluation of policy statements to generate an authorization response. A "default deny" strategy provides safe defaults in case of errors or unexpected situations, such as all policy statements producing a *not applicable* result.

[1]The functions *isSubset*, *intersection* and *union* remove duplicate entries from its parameters before producing its result.

We chose the "deny-overrides" algorithm, described in the XACML standard [22, pp. 133–134], to combine policy statement decisions.

The result is "deny" when any single policy statement evaluates to "deny" or "indeterminate"; else, the result is "permit" when at least one statement evaluates to "permit"; otherwise, the result is "not applicable" as all statements are "not applicable." The PDP is encouraged to interpret a "not applicable" result as a "deny". The PDP also collects all obligations produced by the statements. Only obligations from statements deemed applicable, and whose result is the same as the authorization response are kept.

As a convenience, a separate group of policy statements are set up to have no influence on the overall decision, but contribute its obligations to the pool of obligations to execute. These statements are similarly evaluated for applicability, and contribute obligations only if the evaluation result matches the authorization response.

As defined, the language allows the expression of obligations. These specify actions the PEP must execute as a result of allowing or preventing actions in the enterprise. Obligations are collected from policy statements that contribute to the overall decision and a special set of statements that do not influence the overall authorization decision, but contribute obligations to the authorization response.

VI. Conclusions

SCADA systems offer specialized facilities to integrate distributed control systems into centralized interfaces operated by trained personnel. Increasing interconnection of these systems with external networks, mixed use of current and legacy technology and traditional unawareness of computer security issues motivate the creation of new security solutions tailored for these systems.

We created a policy language for the definition of policy statements. These policy statements allow the implementation of an authorization policy tailored to a SCADA system, where each policy statement provides a condition for either the authorization or the denial of a SCADA message. The proposed policy language for SCADA systems will be used to better define a policy evaluation architecture that can be used for compliance evaluation or authorization enforcement. Our plan is to ensure that such an architecture, based on the concept of Policy Enforcement Points (PEP), allows the composition of these PEPs into hierarchical structures that follow the topology of the SCADA network.

References

[1] American Gas Association, Cryptographic Protection of SCADA Communications Part 1: Background, Policies and Test Plan, Technical Report AGA Report No. 12 (Part 1), Draft 5, American Gas Association, April 2005.

[2] American Gas Association, Cryptographic Protection of SCADA Communications; Part 2: Retrofit Link Encryption for Asynchronous Serial Communications, Technical Report AGA Report No. 12 (Part 2), Draft, American Gas Association, November 2005.

[3] Scott Barman, *Writing Information Security Policies*, New Riders, Indiana, November 2001.

[4] Karl Best, OASIS TC Call for Participation: XACML, OASIS XACML Mailing List (http://lists.oasis-open.org/archives/xacml/200104/msg00000.html), April 2001.

[5] Stuart A. Boyer, *SCADA: Supervisory Control and Data Acquisition*, Third Edition, ISA – Instrumentation, Systems and Automation Society, 2004.
[6] British Columbia Institute of Technology (BCIT), Good Practice Guide on Firewall Deployment for SCADA and Process Control Networks, Technical Report, National Infrastructure Security Coordination Centre (NISCC), London, United Kingdom, February 2005.
[7] CAN in Automation, CAN in Automation (CiA): Controller Area Network (CAN) (http://www.can-cia.org/), November 2008.
[8] Emerson Process Management, Network 3000 Communications Application Programmers Reference, Technical Report D4052, Emerson Process Management, Watertown, Connecticut, USA, October 2007.
[9] Emerson Process Management, ROC Protocol User Manual, Bulletin A4199, Emerson Process Management, Houston, Texas, USA, June 2007.
[10] IEC, Communication Networks and Systems in Substations, IEC 61850-SER, IEC, August 2007.
[11] IEC, Power Systems Management and Associated Information Exchange – Data and Communications Security, Part 1: Communication Network and System Security – Introduction to Security Issues, IEC TS 62351-5, IEC, May 2007.
[12] Innominate Security Technologies AG, Industrial IT Security With Firewall and VPN Hardware – Home – Innominate (http://www.innominate.com), November 2008.
[13] Instrumentation Systems and Automation (ISA) Society, Enterprise–Control System Integration Part 1: Models and Terminology, Technical Report ANSI/ISA-95.00.01-2000, American National Standards Institute (ANSI), July 2000.
[14] Instrumentation Systems and Automation (ISA) Society, Security for Industrial Automation and Control Systems Part 1: Terminology, Concepts and Models, Technical Report ANSI/ISA-TR99.00.01-2007, American National Standards Institute (ANSI), 2007.
[15] Instrumentation Systems and Automation (ISA) Society, Security Technologies for Industrial Automation and Control Systems, Technical Report ANSI/ISA-TR99.00.01-2007. American National Standards Institute (ANSI), 2007.
[16] Merriam-Webster, Policy, in *Merriam-Webster Online* (http://www.merriam-webster.com/dictionary/policy), July 2008.
[17] Modbus IDA, Modbus Application Protocol Specification (http://www.modbus.org/specs.php), April 2004.
[18] Modbus IDA, Modbus Messaging on TCP/IP Implementation Guide (http://www.modbus.org/specs.php), June 2004.
[19] Modbus-IDA, Modbus-IDA: the Architecture for Distributed Automation (http://www.modbus.org/), October 2008.
[20] Modbus-IDA, Modbus Over Serial Line Specification – Implementation Guide (http://www.modbus.org/specs.php), February 2002.
[21] OASIS, eXtensible Access Control Markup Language XACML Version 1.0, Technical Report, OASIS, February 2003.
[22] OASIS. eXtensible Access Control Markup Language XACML version 2.0, Technical Report, OASIS, February 2005.
[23] Organization for the Advancement of Structured Information Standards (OASIS), Oasis Foundation Web Page (http://www.oasis-open.org/home/index.php), June 2008.
[24] Jon Postel, Transmission Control Protocol, RFC 793 (Standard), September 1981.
[25] Hal Stern, *Managing NFS and NIS*, O'Reilly and Associates, Inc., Sebastopol, California, USA, 2001.
[26] Keith Stouffer, Joe Falco and Karen Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, Final Public Draft, NIST, September 2008.
[27] Sun Microsystems, Sun's XACML implementation (http://sunxacml.sourceforge.net/), November 2008.
[28] Mike Thesing, Transporting DNP V3.00 Over Local and Wide Area Networks, Technical Report, DNP Users Group, December 1993.
[29] Mike Thesing, DNP3 Specification Volume 7: IP Networking, Technical Report, DNP Users Group, 1998.
[30] Xin Wang, Guillermo Lao, Thomas DeMartini, Hari Reddy, Mai Nguyen and Edgar Valenzuela, XrML – eXtensible Rights Markup Language, in *XMLSEC '02: Proceedings of the 2002 ACM workshop on XML security*, ACM, New York, New York, USA, pp. 71–79, 2002.
[31] Andrea Westerinen, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Shai Herzog, An-Ny Huynh, Mark Carlson, Jay Perry and Steve Waldbusser, Terminology for Policy-Based Management, RFC 3198, November 2001.
[32] Kurt D. Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, RFC 4510, June 2006.

Rodrigo Chandía He received his Ph.D. in Computer Science at the University of Tulsa in 2009. Dr. Chandia currently works as a Software Engineer at Google, Inc. perfecting the Google Web Toolkit. His areas of interest are: agile software development, compilers, network security and computer security.

Mauricio Papa He received his M.S. (Electrical Engineering) and Ph.D. (Computer Science) degrees at The University of Tulsa in 1996 and 2001 respectively. Dr. Papa is an Associate Professor of Computer Science and the Director of the Institute for Information Security (iSec) at The University of Tulsa. His areas of interest are: process control systems security, network security and distributed systems.