RBAC Driven Least Privilege Architecture For Control Systems

Final Technical Report

23-Jan-14

Work Performed Under Agreement
DE-OE0000544

Submitted By:
Honeywell International Inc., Honeywell ACS Labs (Prime Recipient)
Automation and Control Solutions
1985 Douglas Drive North
Golden Valley, MN 55422

Honeywell Program Manager/Principal Investigator
Julie Hull / Tom Markham
Phone: 763-954-6921 / 763-954-6840
Emails: julie.hull@honeywell.com
tom.markham@honeywell.com

Submitted To:
U.S. Department of Energy
National Energy Technology Laboratory
Diane Hooie
Phone: 304-285-4524
Email: Diane.Hooie@NETL.DOE.GOV

Team Members
Honeywell Process Solutions
University of Illinois at Urbana Champaign (UIUC) – Information Trust Institute
Idaho National Labs

Table of Contents

Table of Figures

1 Executive Summary

The concept of role based access control (RBAC) within the IT environment has been studied by researchers and was supported by NIST (circa 1992). This earlier work highlighted the benefits of RBAC which include reduced administrative workload and policies which are easier to analyze and apply. The goals of this research were to expand the application of RBAC in the following ways.

- Apply RBAC to the control systems environment: The typical RBAC model within the IT environment is used to control a user's access to files. Within the control system environment files are replaced with measurement (e.g., temperature) and control (e.g. valve) points organized as a hierarchy of control assets (e.g. a boiler, compressor, refinery unit). Control points have parameters (e.g., high alarm limit, set point, etc.) associated with them. The RBAC model is extended to support access to points and their parameters based upon roles while at the same time allowing permissions for the points to be defined at the asset level or point level directly. In addition, centralized policy administration with distributed access enforcement mechanisms was developed to support the distributed architecture of distributed control systems and SCADA.
- Extend the RBAC model to include access control for software and devices: The established RBAC approach is to assign users to roles. This work extends that notion by first breaking the control system down into three layers 1) users, 2) software and 3) devices. An RBAC model is then created for each of these three layers. The result is that RBAC can be used to define machine-to-machine policy enforced via the IP security (IPsec) protocol. This highlights the potential to use RBAC for machine-to-machine connectivity within the internet of things.
- Enable dynamic policy based upon the operating mode of the system: The IT environment is generally static with respect to policy. However, large cyber physical systems such as industrial controls have various operating modes (start-up, normal operation, emergency, shut-down and maintenance are typical). The policy enforcement architecture must be able to support changes in access permissions as the mode of the control system changes. For example an operator's role may not allow the operator to shut down a pump during "normal operation" but that same operator role may be given permission to shut down the pump if the refinery transitions to "emergency" mode.

The effectiveness of the approach was validated by applying it to the Experion Process Knowledge System. This is a large commercial industrial control system often used to control oil refineries and other assets within the oil and gas sector. As a by-product, other industries using Experion (Pharmaceuticals, Specialty Chemicals, etc.) also benefit from increased security. Policies representative of those that would be used within an oil refinery were created and validated against the RBAC model as implemented in the underlying SQL database. The administration of policy is simplified which in turn makes it practical for security administrators to specify policies which enforce least privilege. The result is a qualitative reduction in risk. The benefits of the enhanced RBAC model are clear and as a result, Honeywell is incorporating portions of the RBAC research into the 2014 release of Experion.

2 Introduction

2.1 Goals and Objectives

The objective of this project was to create and commercialize a role-based access control (RBAC) - driven least privilege architecture for control systems. The supporting objectives for achievement include:

- *Define a least privilege DCS architecture***.** The distributed control systems (DCS) architecture supporting the specification and enforcement of least privilege was defined and documented in the architecture specification – Role Based Access Control Software Architecture Description, Version 1.1, Dec 2011. The conceptual model is shown in Figure 1. The architecture applies user/application layer proxies and IPsec VPNs as a part of the policy enforcement suite. The system uses existing enterprise authentication server technology to authenticate users. The RBAC policy model then determines the set of least privileges for the subject in context. The RBAC function also drives automated key management on encryption devices/functions. The end state architecture supports network layer encryption. Early in the project the use of application layer encryption was discussed. However, analysis showed that this was not required if IPsec is used so that aspect of the architecture was eliminated.
- *Create the transition approach***.** The DCS infrastructure cannot be changed overnight. Once the end state vision was established, Honeywell developed a transition plan that provides the path for retrofitting security for legacy devices. The plan moves the RBAC policy enforcement point into field devices as the components within the infrastructure are replaced. Legacy devices that support only an all-or-nothing model will be augmented with an application layer proxy in a bump in the wire (BITW) device during the long transition. The set of privileges specified via RBAC will be automatically mapped to an application layer proxy/user rights configuration.
- *Implement, test and demonstrate***.** The project implemented an instance of the architecture as a proof of concept within the Honeywell Experion DCS product. The architecture, design, and implementation have been subjected to security review by the cyber security team at INL. The results have been demonstrated to DOE and shown at the Honeywell User Group (HUG) conference.
- *Commercialization*. Honeywell and our teammates have made a significant cost share contribution as part of a larger plan to integrate the results into existing products and services. The Honeywell Experion® DCS product, developed by Honeywell Process Solutions, is sold to oil and gas operations critical to our nation’s energy. Honeywell Process Solutions played a significant role in the commercialization of the RBAC technology. This included support from marketing and product management, demonstration of the technology at the Honeywell User Group conference and the engineering team.

Goals

The goals for organizations adopting the RBAC technology included:

- *Improved password management:* Provide an integrated set of tools that works with existing enterprise authentication to allow organizations to use the RBAC solution as a front end to legacy SCADA and DCS, allowing better password management and strengthening the first line of defense. The Experion implementation is integrated with a lightweight directory access protocol (LDAP) server which allows users to use their enterprise password to authenticate access to field devices.
- *Simplified security administration*: Providing a control system-centric RBAC model simplifies assignment of user rights. The asset hierarchy of the control system is integrated into the RBAC policy model. This reduces administrative workload and associated errors. The administrative tasks associated with key management are also simplified by relying on automated Public Key Infrastructure (PKI) based IPsec key management.
- *More rigorous privilege definition and enforcement:* Providing the ability to rigorously define roles and operating modes enables organizations to establish and adhere to operating procedures that enforce safety constraints and reduce undocumented changes to process parameters. For example, view-only operators assigned to a specific plant area can observe the state of that area of the plant but not make unauthorized changes.

2.2 Project Summary

The hypothesis of this work was that by adapting the RBAC concepts to the unique characteristics of the control system environment the result would be a cost effective approach for implementing least privilege in SCADA and DCS. The security community has long known that the impact of a security breach (e.g. hostile insider, subverted software) can be minimized by providing subjects (users, software and devices) only the set of permissions necessary to do their job. The challenge is that previous security policy mechanisms were very time consuming to configure and difficult to analyze. Thus, even though security administrators knew the value of least privilege they found that the cost of implementation was simply too high.

The overall approach for the effort was to combine several concepts within the context of control systems. These concepts included:

- Layered policy and enforcement
- Permission inheritance via an asset hierarchy
- Permission assignment via parameter types
- Insertion of policy enforcement points as near as practical to the resource to be protected.

The conceptual model of the RBAC components which were overlaid upon Experion is shown below in Figure 1. The components of the model are:

- Policy Decision Point (PDP A.K.A. policy server): This contains the policy database and engine for converting the policy specification into access vectors.

- Certificate Authority (CA): This serves as the certificate authority for a local system. It provides signed certificates to support devices securely joining the network and for establishing IPsec communications.
- Policy information point: This represents an auxiliary source of data which may be used within the policy decision point to make access control decisions. Typical information which might be provided is employee safety training records. If an employee’s safety training is not current, the employee could be denied access to specific resources.
- Policy administration point: This represents the interface used by the administrator for adding users and creating or modifying policy.
- Policy audit and analysis: This represents the interface and functions which an auditor would use to perform analysis of policy (e.g. are there orphan devices?) and track changes to the policy.
- Communications policy enforcement: This represents the IPsec/ Internet key exchange (IKE) implementation which is actually distributed between pairs of securely communicating nodes. The node to node access vector is interpreted on the device and used to drive the IKE negotiation. IPsec is then used to secure the traffic between nodes.
- Data client: The client represents a node (e.g., an operator workstation) which exchanges application layer data with a data server.
- Policy agent: This represents a function on a data client which assists in authenticating the requestor (node, application or user) to the policy enforcement point.
- Data server: This represents an entity which serves data or control functions to a requestor. It may be a control server which caches data from many control points. It could also be a field controller (e.g. C300) which provides direct access to control points.
- Application policy enforcement point: This function receives an access vector from the policy decision point, authenticates an incoming request and applies the access policy to the request.

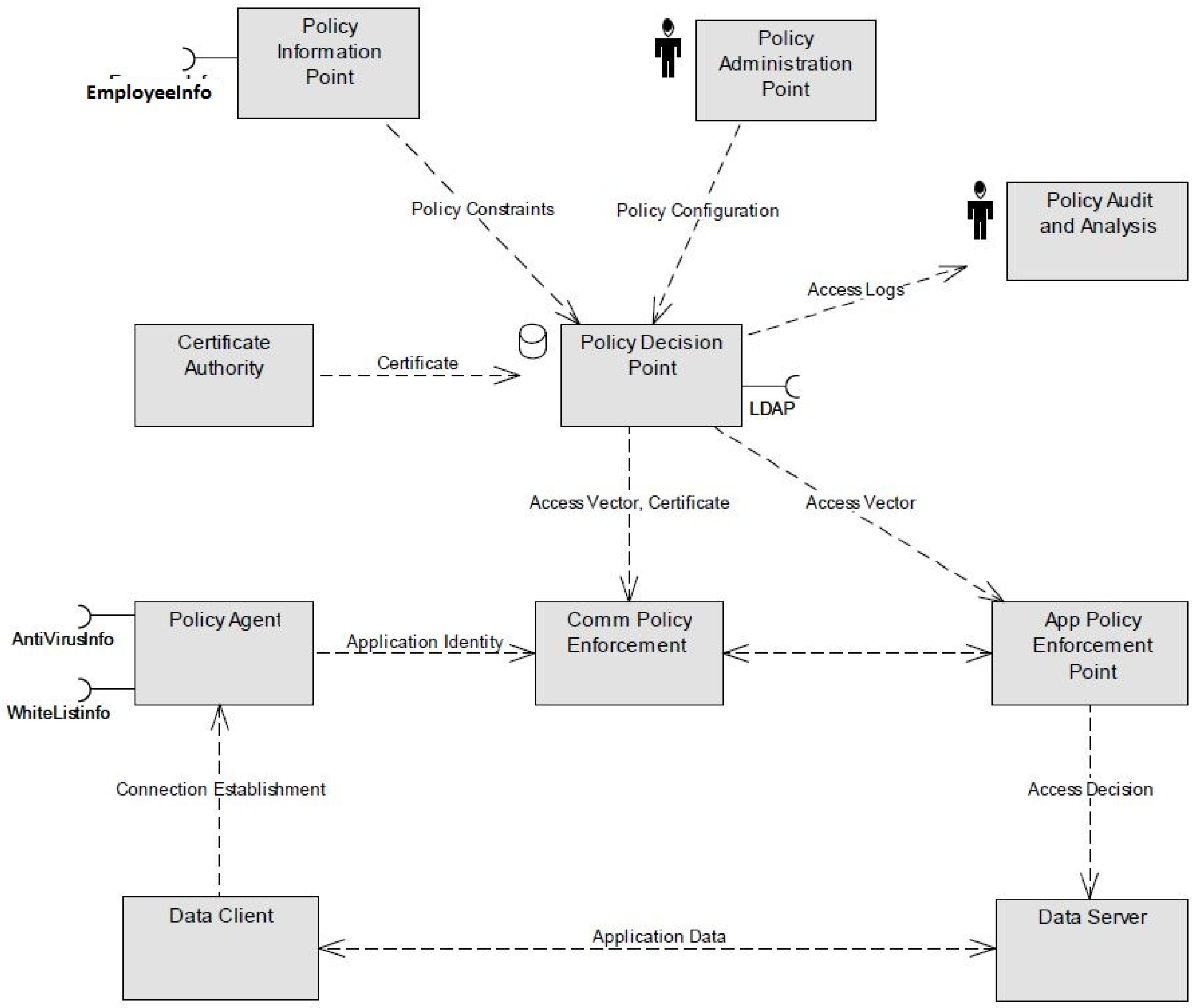


Figure 1 The conceptual model of the RBAC system components.

The concept of layered policy enforcement is that policy can be simplified and overall security enhanced by leveraging the natural layers in the system and creating policy enforcement points tailored to the characteristics of the layers. The approach used three layers as shown below in Figure 2.

The lowest layer is the node to node communications policy. The principle of least privilege implies that not every node should be able to communicate with every other node. By limiting which nodes are allowed to communicate, entire classes of threats can be mitigated. The number of nodes in a control system is generally orders of magnitude smaller than the number of points and parameters. Thus, creating policy is relatively simple. The use of IPsec as the enforcement mechanism (policy enforcement point) results in access enforcement which is robust and difficult for an attacker to bypass.

The middle layer provides control over which software can access a remote service or application. The software may be authenticated (weakly) by its pathname. However, the preferred method is to have the software identified/authenticated by a commercially available white listing product and then use RBAC policy enforcement to control the access the software is given.

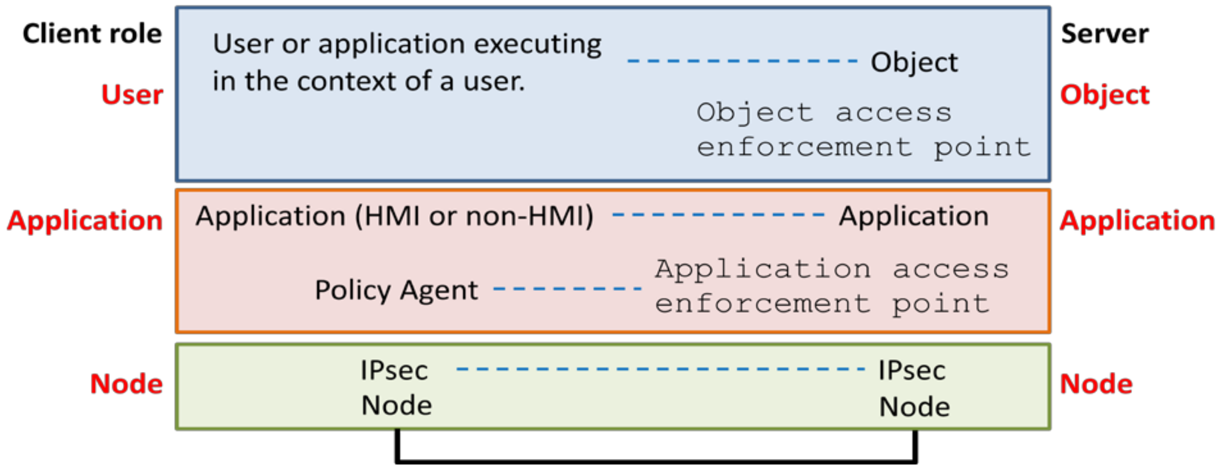
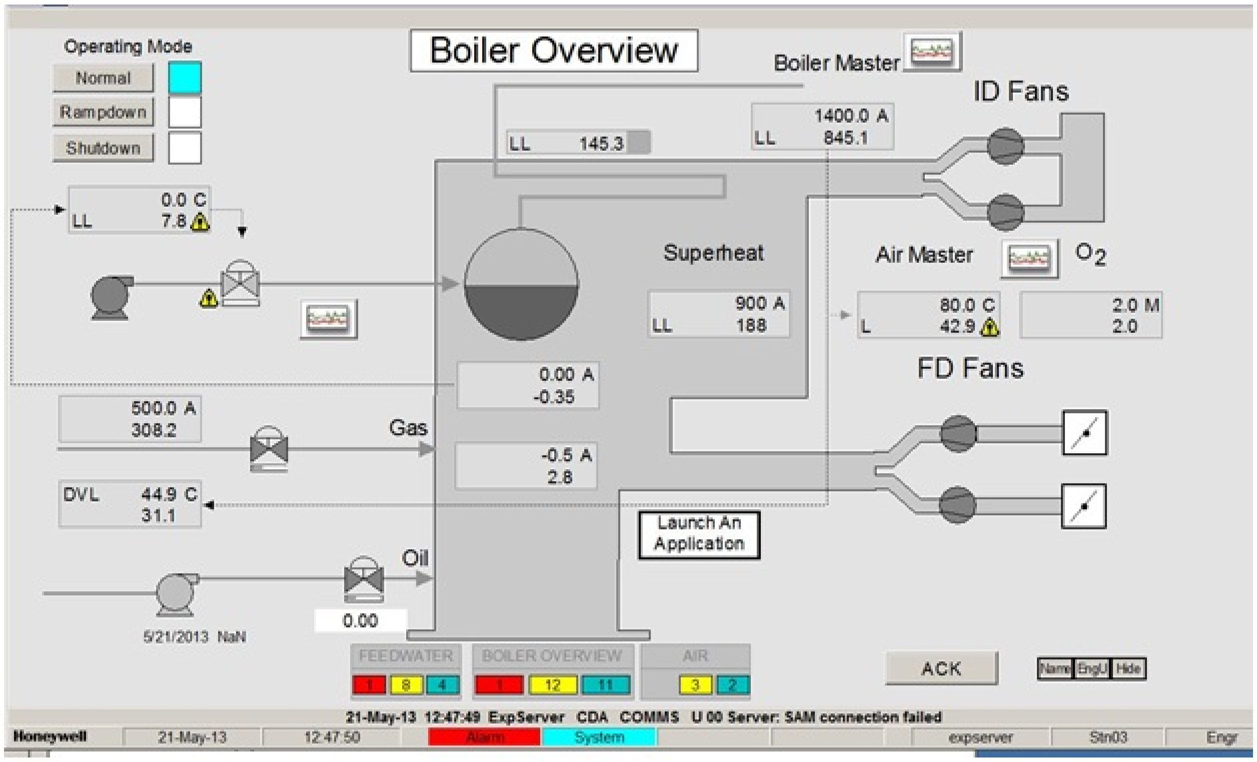

Client role
User
User or application executing in the context of a user.
Object
Object access enforcement point
Server
Object
Application
Application (HMI or non-HMI)
Application
Policy Agent
Application access enforcement point
Application
Node
IPsec Node
IPsec Node
Node

Figure 2 The RBAC layered enforcement model

The upper layer enforces the user role access to security objects which are typically points and parameters within the control environment. Within control systems the number of user roles is relatively small (typically much less than 100) while the number of security objects may be tens or hundreds of thousands. This scale is addressed by applying an asset hierarchy and parameter types as described below.

The asset hierarchy provides a logical grouping of the points within the control system and thus allows the security administrator to assign permissions to the related points quickly and easily. The boiler shown in Figure 3 represents a typical plant asset. The security administrator may assign the operator role access to the boiler and the operator inherits permission to all of the points (pumps, valves, temperature set points, etc.) assigned to the boiler asset. The administrator is able to modify permission to a specific point through the use of *Exception* and *Constraint* mechanisms.



21-May-13 12:47:49 ExpServer CDA COMMS U 00 Server: SAM connection failed

Figure 3 The boiler represents an asset with many points under it

Points within a control system have parameters associated with them. Typical parameters types are

- Set point: The target value for the physical property being controlled. E.g. hold the temperature at 220 degrees.
- Current value: The current value for the physical property (215 degrees)
- High alarm limit: Trigger an alarm if the value rises above this limit.
- Low alarm limit: Trigger an alarm if the value drops below this limit.

The example above illustrates that for a single point (boiler temperature) there may be multiple parameters. The safe operation of a DCS implies that access to these parameters should be different for the various roles. For the example above, the roles and their accesses could be as follows.

- View only operator: This role is able to view the operation of the plant but not change anything. This role would be given view only (read) access to all 4 parameters associated with the boiler temperature point.
- Operator: The operator is responsible for day to day operation of the boiler so the operator role is given write access to the set point. The operator has read only access to the other parameters.
- Engineer: The engineer is responsible for safe operation of the plant but not day-to-day production. The engineer is given the ability to set (write) the high alarm limit and low alarm limit but the role is only given read access to other parameters.

The discussion above highlights the fact that a role may have access to a point but the permission to access the parameters is a function of the role. Thus, within the DCS environment the RBAC model is modified to provide two sets of orthogonal access permissions which are then ANDed together. This unique aspect of orthogonal access permissions shaped the planned tasks, and in particular the policy model research described in Section 4 - Access Control Model.

2.3 Planned Tasks

The work was broken down into three major phases.

Research

This focused on defining the problem and outlining the theoretical work necessary to advance the state of the art. The research phase goal was to produce documentation for secure remote access control architecture for control systems using an RBAC model to identify and enforce least privilege. A transition plan for a moving toward the secure architecture, recognizing the long lifetimes of control system equipment, was also documented. The team developed requirements and an architecture by exploring both uses cases and the threats to the system. The requirements highlighted the need to extend existing access control models to address the control system environment. The Information Trust Institute (ITI) at the University of Illinois led the research regarding policy models.

Detailed Design and Development

This focused on taking the research results, requirements and architecture defined during the research phase and translating them into running code in the context of the Experion system. Honeywell Labs worked very closely with the Honeywell Process Solutions (HPS) business unit to support insertion of a subset of the RBAC technology into the Experion RBAC prototype. This implementation includes:

- User interface tools for specifying RBAC policy.
- Interface mechanisms to an LDAP enterprise authentication server to support RBAC.
- A Bump In the Wire (BITW) proxy function which applies user rights management to remote access attempts to a legacy field device.
- A certification authority to support RBAC driven key management.
- Embedded cryptography, based upon IPsec, for secure communications between nodes.

The functions above were integrated to produce a prototype Experion based system capable of demonstrating the application of RBAC to provide least privilege.

Test, Evaluation and Demonstration

This phase delivered the prototype system to Idaho National Labs (INL) for security testing by their red team. Demonstrations of the system were performed for DOE as well as Honeywell leadership in order to stimulate commercial adoption of the technology. Honeywell is commercializing the technology.

The specific tasks planned and executed for each phase of the program are introduced below.

Research

Task #	Title	Description
1.01	Project kick off	Project start-up tasks including setting up the collaboration environment, configuration management repository, and updating the literature search.
1.02	Least privilege architecture	Developed and documented the RBAC-driven least privilege architecture. The output is the end-state vision.
1.03	Requirements	The requirements for the system functions were developed in parallel with the architecture. These requirements include those for the RBAC enhanced authentication/authorization server, RBAC policy specification, continuous validation key server (certification authority), proxy server (BITW) and embedded cryptography (IPsec).
1.04	Network design and migration	Developed and documented the network design and the migration plan for transitioning from the existing insecure control system environment to one supporting the RBAC-driven least privilege architecture.
1.05	Phase 1 Reviews	Provided support for all reviews internal to the project team as well as those with the customer.
1.06	Phase 1 Program management	Program management provided for all management aspects of the research phase including cost, schedule and performance tracking. The first subtask under the program management task was to update the project plan based upon contract negotiations.
1.07	UIUC Phase 1	Work done at UIUC. The initial subtasks included Real time RBAC with continuous validation, RBAC learning, review of the architecture and migration plan, participation in team meetings/reviews and program management for the UIUC portion of the effort. These were adjusted as the requirement and architecture matured.
1.08	INL Phase 1	Work to be done at INL. The subtasks included leading a cyber security review of the architecture, review of the migration plan, and a review of the continuous validation approach (IPsec). INL also participated in program reviews.

Development

Task #	Title	Description
2.01	Authorization server	Detailed design and implementation of the RBAC authorization server (A.K.A. Policy Distribution Point (PDP)) integrated with an enterprise authentication server.
2.02	RBAC key server	Detailed design and implementation for the RBAC key server (A.K.A. Certification Authority). This work was initially to extend the existing Honeywell One Wireless key server. However, the certificate authority based design was selected because it is more scalable.

2.03	Proxy framework	Detailed design and implementation for the RBAC proxy that will be the policy enforcement point for legacy control devices that cannot enforce fine-grained user access rights. This proxy was implemented as the BITW device which provides access control at the device, application and user levels.
2.04	Embedded cryptography	Cryptographic functions in a device or software function. The team explored working with Topic Area 5 – Secure Communications team to jointly implement their solution in a device or function. However, the RBAC model required dynamic connection management to provide timely enforcement of policy changes. Therefore, the team selected an IPsec/IKE approach.
2.05	Network system integration	Integration of the technologies above as well as the UIUC development. The result is a functional system that demonstrates the end-to-end RBAC solution on a commercial product.
2.06	Phase 2 Reviews	Support for all reviews both internal to the project team and with the customer.
2.07	Phase 2 Program management	All management aspects of the research phase including cost, schedule and performance tracking.
2.08	UIUC Phase 2	Work to be done at UIUC. The Phase 2 subtasks included RBAC user interface and engine, design and development support for the RBAC key management, reviews of the Authorization Server, Proxy Framework, and Application Layer Cryptography. (The application layer cryptography task was dropped because IPsec met the requirements.) In addition, UIUC participated in reviews and made design and implementation updates driven by output of the INL security reviews.
2.09	INL Phase 2	Work to be done at INL. The subtasks included a security design review and a complete system security review of the resulting design integrated into a Honeywell control system. INL also participated in program reviews

Test, Evaluation and Demonstration

Task #	Title	Description
3.01	Demonstration plan and implementation	This task provided for the development, analysis and implementation of the demonstration plan.
3.02	Commercialization plan	This task developed the commercialization plan for taking the RBAC technology to market. The plan takes advantage of the existing Honeywell Experion product which is already deployed in the electric, oil and gas industries.
3.03	Phase 3 reviews	This task provided support for all reviews internal to the project team as well as those with the customer. The demonstration and final project review were addressed within this task.
3.04	Phase 3 Program management	All management aspects of the research phase including cost, schedule, and performance tracking.

3.05	UIUC Phase 3	Work to be done at UIUC. The Phase 3 effort focused on support of the demonstration and commercialization tasks.
3.06	INL Phase 3	Work done at INL. The INL security team performed security testing and provided a report of the results.

2.4 Problems and Challenges

The significant research challenges worthy of further consideration by DOE are:

Integration of safety and security: Safety frequently requires a *fail open* model. E.g. If there is an emergency within a refinery, the security controls should open thus, allowing the operator access to any controls necessary to save lives and minimize damage to the plant. In contrast, security controls generally favor a *fail closed* policy. i.e. If the subject cannot prove they are authorized access, then access is denied. The short term solution to this will be to allow customers to configure their RBAC either for a fail closed or fail open policy. However, research is required to create integrated safety and security risk models as well as create mechanisms which reduce or eliminate the either/or conflict between security and safety.

Usability of security systems: One of the barriers to the adoption of least privilege in the real world is the added complexity, time and cost associated with creating, analyzing and maintaining fine grained access control policies. This project made significant progress on this front by integrating the asset hierarchy and the parameter constraints with the policy model. However, as an industry, more needs to be done to provide a unified and simplified means of specifying least privilege policies in complex, safety critical environments.

IPsec interoperability: IPsec has been a standard for over a decade but interoperability problems still exist. Defining a clear IPsec profile, which we did, using only a small subset of all possible IPsec variations is critical to achieving interoperability. An example of the interoperability challenges that remain is that we were not able to use IKEv2 because to this date it is not supported by Microsoft in transport mode. This is not so much a research issue but a problem which needs to be worked within the standards community.

The significant deviations from the original plan and the impacts are:

- **Continuous validation**: The UIUC team had originally planned to do work in the area of continuous validation. Early thoughts were that some mechanisms would be in place to continuously validate that a subject (e.g. user) was who they claimed to be and that they were still authorized access. However, as the design emerged the team concluded that IPsec together with TCP provided packet and stream integrity which provided continuous authentication across the network. The access revocation mechanism, which could be triggered by a policy change (e.g. removing a user from the LDAP directory or changing permissions), ensured that policy changes were quickly propagated. Therefore, continuous validation as a research task was addressed without significant expenditure of resources.

- **Policy learning**: One of the original research goals was to simplify policy administration by allowing the RBAC system to monitor network traffic and learn policy from existing (functioning) control systems. However, the team quickly learned that many aspects of the policy, though very important, are exercised infrequently. For example, an operator may only access controls for some valves and pumps during an emergency situation. Since emergencies are rare, it is not practical for the RBAC system to learn which policy should be enforced during an emergency. Therefore the team was forced to move toward an approach in which the administrator manually specifies the access rights of various roles. The team was able to reduce the administrative burden by leveraging existing roles, the Experion asset hierarchy and other structure within the control systems.
- **Lemnos**: The team originally hoped to use the Lemnos device (Schweitzer SEL-3620) as the mechanism for providing secure communications. However, discussions with SEL revealed that the device did not have an application programming interface (API) which supported dynamic policy management. i.e., there was no way for the RBAC Policy Distribution Point (A.K.A. policy server) to push new IPsec policy to the SEL-3620 without direct human intervention. Thus, the team was forced to use IPsec and use the RBAC access vector distribution mechanism to distribute IPsec polices to nodes in near real time. The team was still able to leverage a vast amount of work done by the security community to develop and deploy IPsec. One noteworthy feature of Lemnos is that it provides local user authentication while the RBAC BITW device uses the LDAP server. The Lemnos model is appropriate for SCADA environments in which communications to the LDAP server may not be available.

3 Products and Technology Transition

The transition of the technology involves multiple aspects.

- Listening to the customer and incorporating their requirements and feedback into the technology development
- Implementing technology within a commercial product
- Communicating technology to the research and user community
- Protection of Intellectual Property

Customer Focus
Honeywell has obtained Experion specific customer requirements, reviews and feedback throughout the program to guide the implementation of the RBAC functions within the Experion product. The benefits to the customer can be found in the section titled “Security Benefits”. The list of customer interactions includes:

- One-on-one interviews with major energy sector customers and Honeywell Specialty Materials for voice of the customer requirements gathering
- One-on-one briefing to a major energy sector customer followed by a design review
- Presented RBAC at the Honeywell User Meeting in Brussels which included major energy sector customers resulting in excellent feedback for the product.
- Briefings to customers at the Honeywell User Group (HUG) meeting in June 2013.

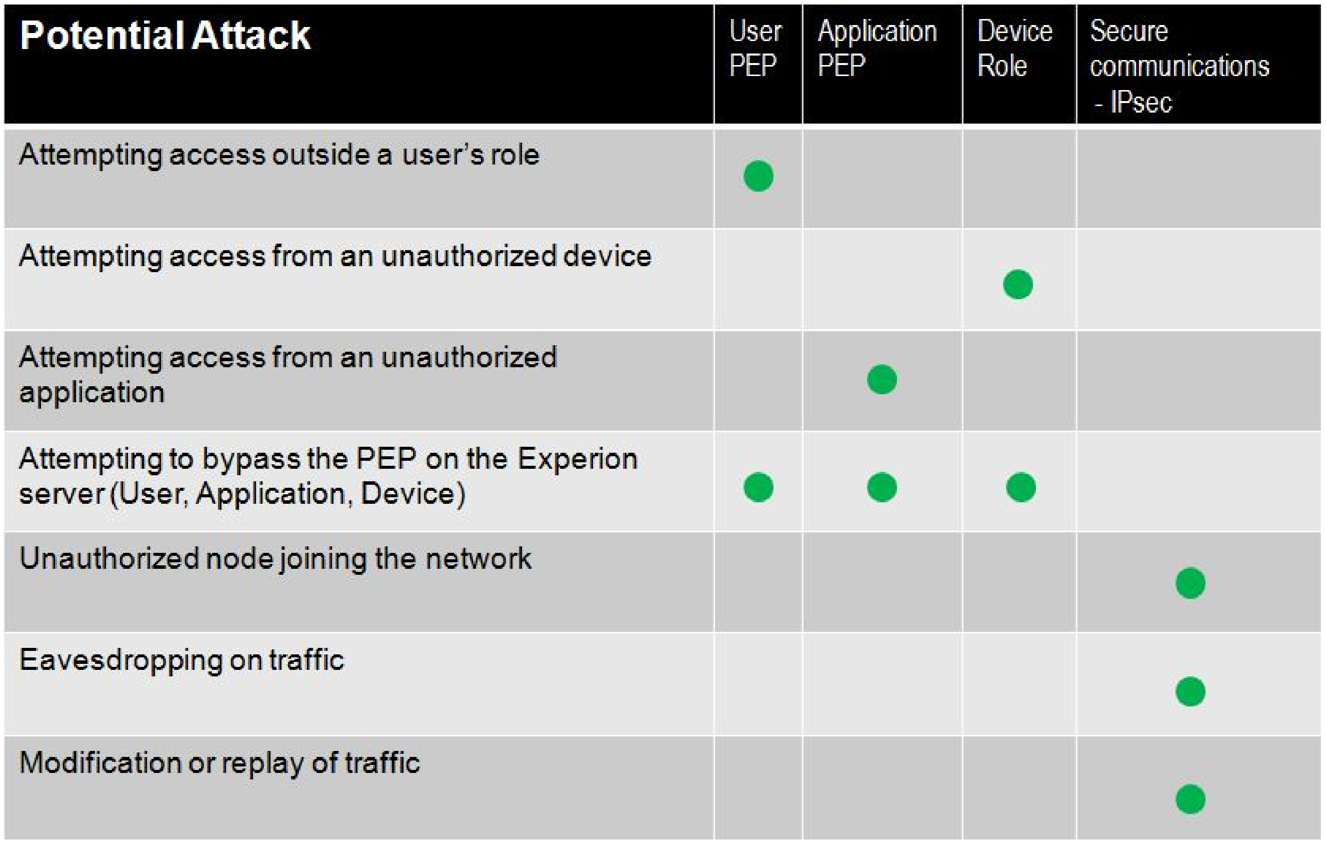
Security Benefits
The incorporation of RBAC into a DCS results in the potential to reduce risks for the oil and gas sectors. Honeywell has communicated these benefits to the research community as well as to Experion customers. Qualitative risk reduction is achieved as products incorporating this technology are incorporated into our energy infrastructure. Figure 4shows classes of high level attacks and the RBAC mechanism which provides effective countermeasures to those attacks.

The list of technologies and techniques which contribute to security and usability includes:

- **Policy model and user interface make fine grained access control policy easy to specify**
    - The policy specification makes use of structure and tools already present within the control system. For example, the existing asset hierarchy allows logical grouping of security permissions. The integration of RBAC with the enterprise authentication system (e.g. Windows Active Directory) allows the administrator to define lists of users in one place. The users may use the same authentication mechanisms including multi-factor authentication for access to the control system and enterprise functions. Thus, administrators are more likely to set a least privilege policy because the workload is manageable.
- **Control is provided at the device, application and user level**

  - Device level control is coarse grained and therefore less burdensome to administer than other fine grained mechanisms. The cryptography used within IPsec (Advanced Encryption Standard) is strong and the IPsec protocols are mature. Application level access control prevents authorized users from using unauthorized applications to access remote resources. Finally the user level access control makes it possible to control access to specific functions or points in field devices as opposed to the all or nothing model of many legacy devices. The result of integrating these technologies is a defense in depth approach which requires many attacks to penetrate multiple layers of security controls.

- **Policy changes are propagated in less than 15 seconds**
  - The policy distribution mechanism actively pushes new access vectors to the policy enforcement points when a policy change occurs. If a device is offline, the change is propagated when the device comes back online. The result is that permission changes are rapidly propagated and the window of vulnerability is reduced.
- **Policy enforcement is pushed to end devices such as the BITW/Modbus and C300 vs. just in the server**
  - Many traditional control systems enforce access permissions at a server near the head end. However, legacy devices in the field lack local policy enforcement mechanisms. As a result, an attacker who is able to bypass the server at the head end is often granted unrestricted access to field devices. The RBAC architecture and implementation places policy enforcement points very close to or within the field devices so that attacks from downstream of the server are reduced.
- **IPsec, Whitelists and LDAP are integrated**
  - Historically many security vulnerabilities occur at the seams within systems. E.g. the set of mechanism results in gaps in the security coverage or the interface to a security mechanism makes invalid assumptions. The integration of IPsec, application whitelisting and LDAP within the RBAC system provides a more (but not completely) seamless security approach. Two secondary benefits of integrating LDAP are:
    - User accounts are not duplicated. i.e., Once within the enterprise and second within the control system.
    - Advanced authentication techniques, such as two factor authentication, which may be supported in LDAP are inherited by the control system.



Potential Attack	User PEP	Application PEP	Device Role	Secure communications - IPsec
Attempting access outside a user's role	●			
Attempting access from an unauthorized device			●	
Attempting access from an unauthorized application		●		
Attempting to bypass the PEP on the Experion server (User, Application, Device)	●	●	●	
Unauthorized node joining the network				●
Eavesdropping on traffic				●
Modification or replay of traffic				●

Figure 4 Potential cyber attacks thwarted by RBAC mechanisms.

The Experion control system environment was used as a base for prototyping the RBAC technology and serves as a vehicle for moving the technology into the marketplace. Experion is an established product already being used in oil, gas and electric power generation. Thus, users do not need to rip out existing products and processes to upgrade to higher security and simplified administration.

The Honeywell C300 controller is a state of the art process controller that can also operate as a programmable logic controller (PLC). The outputs of the C300 controller control the actuators in a process environment. The process sensors interface to the C300 for providing data from measurements such as temperature and flow. The C300 was enhanced to provide RBAC policy enforcement (node, application and user) in the prototype system.

The RBAC technology is being rolled out in the Experion product in stages It is expected that competitive pressure in the market will encourage other control system vendors to match Experion's new security features.

Commercialization

Honeywell has implemented some of the RBAC technology within the Experion product. Honeywell is actively engaging customers to assist in defining future product features. Please see the RBAC Driven Least Privilege Architecture for Control Systems Commercialization Plan for more information.

Outreach

Honeywell has communicated information regarding the research to the community through the following channels.

- The team wrote about the need for cyber security in control systems in the article “Staying in Control” published in the Jan/Feb 2012 edition of the IEEE Power & Energy magazine. The team wrote several papers regarding the technology (e.g., Role-Based Access Control for Distributed Control Systems submitted to IEEE Transactions on Dependable and Secure Computing) which were not accepted. The team has submitted an abstract entitled “Next generation access control for distributed control systems”, to the IEEE Security and Privacy special issue on Energy Sector Control Systems. The team is waiting for feedback on this submission.
- The team provided a briefing on the RBAC technology to the National Electric Sector Cybersecurity Organization (NESCO) in April 2011.
- The team presented a poster at the Honeywell technology symposium in May 2011.
- The team presented an RBAC poster at the Trustworthy Cyber Infrastructure for the Power Grid (TCIPG) workshop in Oct 2012.
- The team presented a briefing and poster at the Cybersecurity for Energy Delivery Systems Peer Review in July 2012.
- UIUC presented a paper titled “A Framework Integrating Attribute-based Policies into Role-Based Access Control” at the ACM Symposium on Access Control Models and Technologies (SACMAT ’12).

Patent Applications

The research resulted in two inventions:

- The application of role based access control within the control system environment, and in particular the late binding aspect is the subject of US patent application 13/682428, “Role-Based Access Control Permissions”, filed on November 20, 2012.
- The use of RBAC concepts to provide automated key management is addressed in the invention disclosure “Policy Based Secure Communication with Automatic Key Management” which was disclosed to DOE in Jan 2013. Honeywell is in the process of preparing a U.S. patent application to be filed within the next few months.

4 Access Control Model

Before discussing the RBAC model developed in this project we provide an overview of a Distributed Control System (DCS, hereafter) that motivated this model and capture its important characteristics and requirements for an access control model. We then describe a concrete RBAC model and policy enforcement framework, tailored for DCS and discuss its performance evaluation. We then describe a formal and more generalized RBAC model that goes beyond meeting DCS requirements and integrates attributes into RBAC.

4.1 Requirements

A DCS is typically organized into multiple hierarchical levels from the highest (Enterprise) level to the lowest (Control module) level, as described in the ISA-88 and ISA-95 standards. An abstracted view of a hierarchical DCS system with four levels is shown in Figure 5. The lower levels are further organized into groups based on the geographic or logical organization of an industrial plant. In the given example, the plant is divided into two “controller zones”, Zone A and Zone B, at the supervisory level. A controller zone groups logical and physical assets that share common controller objectives and requirements. Although not shown in Figure 5, these zones could be further divided based on processes at the lower levels. At the lowest level, Level 1, are controllers that interface with, monitor, and control the physical processes. Each controller runs many control algorithms, also referred to as control blocks or points. There are different types of points or control blocks; PID (Proportional-Integral-Derivate) is one such type. Each point, based on its type, has a set of parameters associated with it that are used by the control algorithm. Examples of parameters include process variables, set point limits, output limits, alarm limits, and alarm-enabled state. At the next level, Level 2, are supervisory devices that allow plant operators, engineers, and managers to monitor and control the physical processes by accessing the controllers. The devices include operator consoles that act as the human machine interfaces (HMI) and data caches that cache sensor data to provide faster access to field device data. The use of the data caches also minimizes the number of data reads/writes performed directly on the controllers, reducing their workload. Level 3 represents operations management and includes dispatching, production scheduling, reliability assurance, plant-wide control, and optimization. At the topmost level, Level 4, is the enterprise system that is used for business planning and logistics.

The RBAC requirements are driven by the following characteristics of the DCS environment.

Large number of security objects and permissions: A typical DCS might have tens of controllers, with each controller having thousands of points, and each point having tens of parameters. Each point and its parameters have multiple available operations (e.g., read, write, and configure). That creates an explosion of security objects and permissions, making their management and assignment tedious and error-prone. Further, access to and allowable operations on a point parameter are limited by an entity’s access to the point itself.

Requirement 1. An RBAC solution must provide compatible and easy means to migrate and capture existing point and parameter objects, and leverage the access relationships between them. Further, it should ease the specification and management of security objects and their permissions.

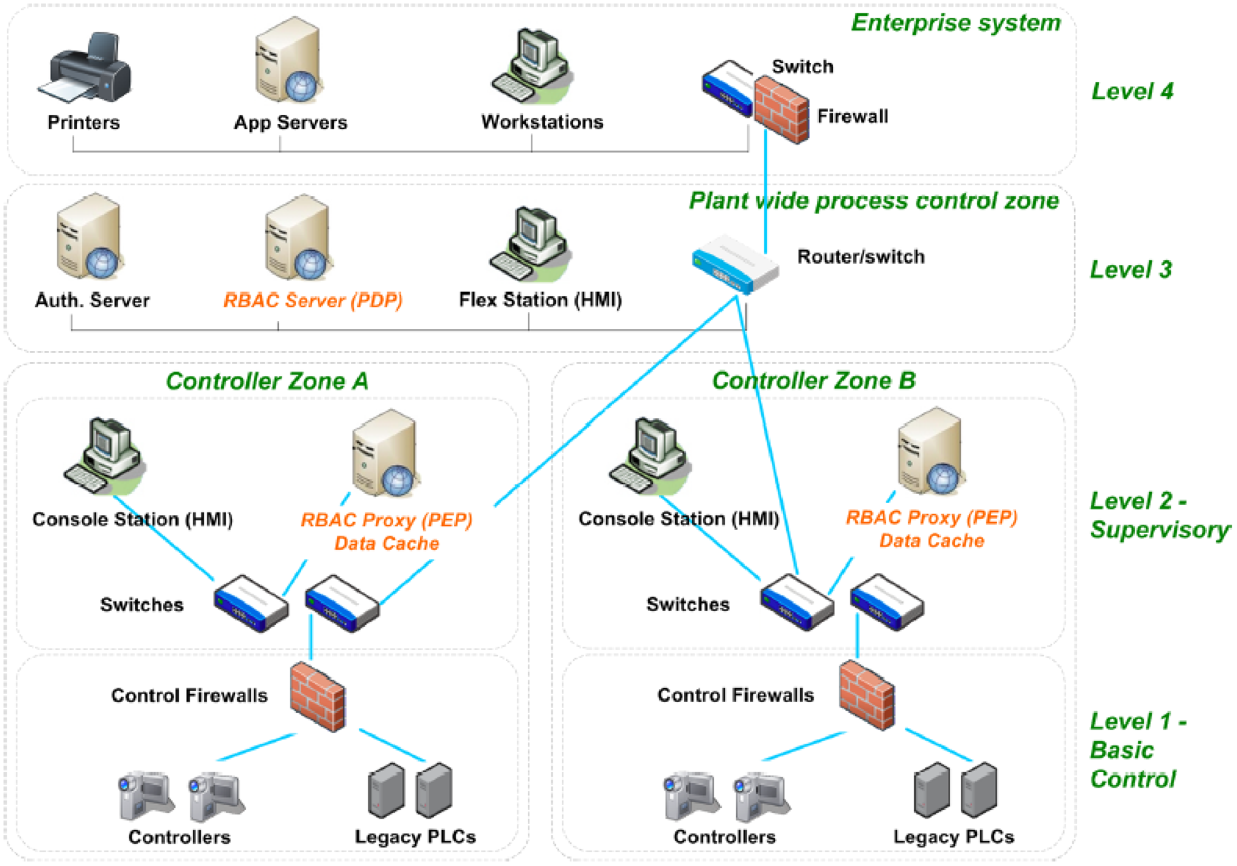


Figure 5 DCS overview with RBAC components

Well-defined job functions and access patterns: DCS and process control systems in general have well defined and relatively stable basic job functions, with well-defined access rights or access right patterns within a system context. For example, operator and engineer are well-understood job functions in the DCS community and are well-defined within a given process control context. An operator is typically only allowed to modify operational parameters, whereas an engineer is typically also allowed to configure control algorithms. Users performing those job-functions are often limited in scope of responsibility to logical control entities (e.g., a cracker) of the plant corresponding to sub-processes.

Further, in a DCS system, access patterns on the parameters are the same for a given job function regardless of the individual points they are associated with. For instance, suppose an operator is given responsibility for monitoring control points A, B, and C, each of which uses a parameter “set point limit (SP)”; then the operator would have the same access rights to the set point limits on all three points, i.e., A.SP, B.SP, and C.SP. The second requirement insists that an RBAC solution exploit the access patterns mentioned above:

Requirement 2. An RBAC solution should leverage the presence of well-defined job functions and associated access right patterns to ease the specification and management of roles and objects.

Hierarchical organization of security objects: DCS systems are typically organized hierarchically, with both logical and physical groupings based on controller objectives, geographic location, or plant organization among other things. A logical grouping of control objects, such as points that correspond to a part of the control system entity, is typically referred to as a control asset. For example, in an oil refinery setting, a "cracker" or a "boiler" can be a large control entity (itself represented as an asset) controlled by several "control loops," each with some number of points. Relationships like that lead one to organize control assets within a tree hierarchy (see Figure 7). A system asset, on the other hand, represents the physical grouping of the actual field devices and server machines, including operator consoles, HMIs, and controllers. These are also organized in a hierarchy, reflecting their positions within the network. An operator monitoring a part or a control entity in the plant typically has access, at a predefined level, to all the control and system assets associated with the control entity, with a few exceptions.

Requirement 3. An RBAC solution should support a hierarchy of object groups and permission inheritance in the hierarchy. Further, it should support multiple exceptions to permission inheritance in the object hierarchy.

Station-based or application-based access: Station based security is a commonly used access model in DCS. Under station-based access control, console stations are preconfigured to have a certain level of access to some assets within the plant, each connected to specific parts of the control system. An operator with access to a particular console station gets all the access rights the console station has. Typically, the zone in which the station is located determines what assets it can access. While station-based access control does not enforce least-privilege, it does capture important access constraints that should be considered when determining the effective access level of an operator. This notion of contextual access control also applies to DCS applications (e.g., an HMI application) that operators use to make requests. An application could have a more restricted level of access to some assets, affecting the effective access level of an operator while requests are being made through it.

Requirement 4. An RBAC solution should support station-based and/or application-based access constraints.

Real-time and safety-critical system: Many DCS systems have real-time and safety requirements. Correspondingly, access control mechanisms should add as little latency as possible to system operations. For example, cross level interactions should be minimized.

Further, a failure in the access control system components should have minimal interference with the way operators perform their daily operations.

Requirement 5. With all of the RBAC mechanisms in place, the additional operation time to complete a control or administrative request, or the additional response time for an operation failure, shall be less than a few hundred milliseconds.

Requirement 6. DCS downtime resulting from a failure in the RBAC solution should be minimized and have minimal interference with the daily operations of the safety critical functions.

4.2 DCS-RBAC Policy Model

To address the various features of DCS, we designed DCS-RBAC model. At a high level, DCS-RBAC

- uses proto-permissions and proto-permission groups instead of traditional permissions that associate an operation with an object;
- uses proto-objects, object groups, and an object group hierarchy; and
- supports exceptions to permission inheritance in the object group hierarchy, among other things.

Different from roles in the standard model, a role in DCS-RBAC associates a group of proto-permissions with a group of objects; a role also associates a Scope of Responsibility, together with proto-permissions to determine the instances of permissions. In addition, standard roles are expanded to application roles and device roles, allowing applications and devices to be assigned to their own roles and have permissions. A high level view of DCS-RBAC policy model is illustrated in Figure 6.

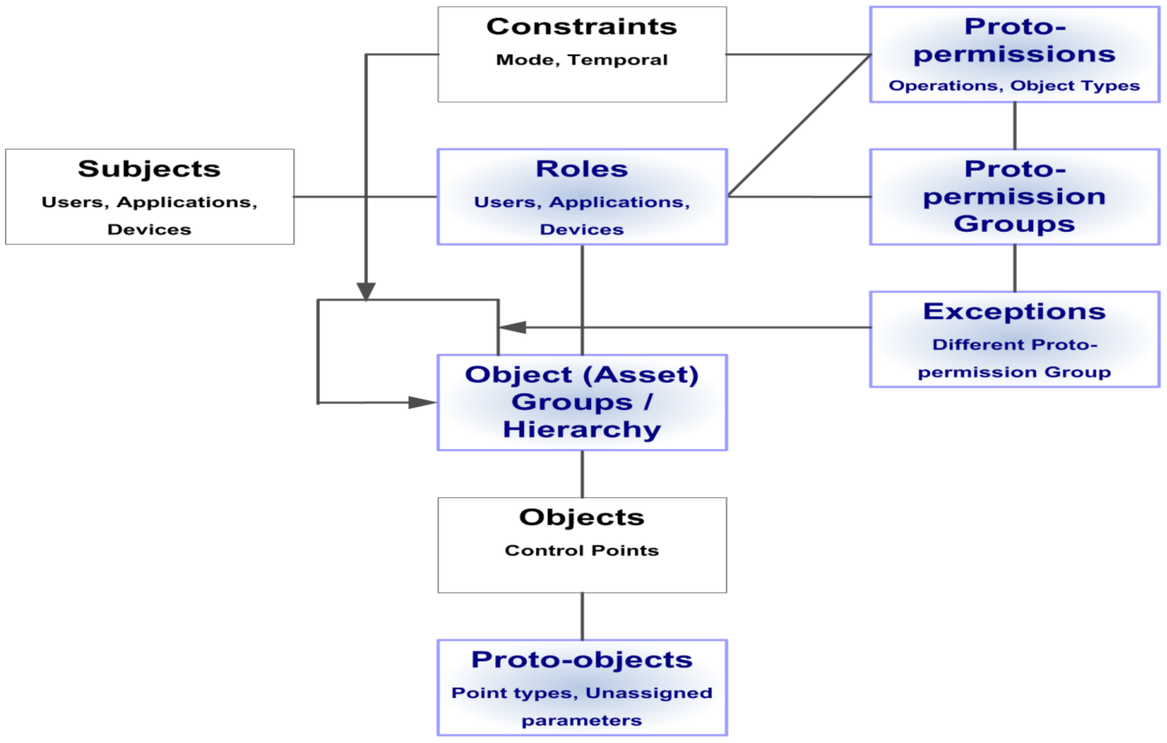

Constraints
Mode, Temporal
Proto-permissions
Operations, Object Types
Subjects
Users, Applications, Devices
Roles
Users, Applications, Devices
Proto-permission Groups
Exceptions
Different Proto-permission Group
Object (Asset) Groups / Hierarchy
Objects
Control Points
Proto-objects
Point types, Unassigned parameters

Figure 6 DCS-RBAC. Blue boxes show our additions/changes to the basic RBAC model.

4.2.1 Key Concepts

Some key concepts are given as follows.

Proto-permissions. A proto-permission pp (id; type; op; objType) associates an operation (*op*) with an object type (*objType*); by contrast, traditional models associate an operation with a specific object instance. Through type, we classify whether a proto-permission is of type "point," "parameter," or "administrative." Below is an example set of proto-permissions for DCS:

pp(1, point, view information, point)

pp(2, parameter, write, PID.SP)

pp(3, administrative, configure settings, system flex station).

In the example above, *pp 1* is a "point permission" that allows the "view information" operation to be
performed on objects of type "point"; *pp 2* is a "parameter permission" that allows the "write" operation to be performed on objects of type "PID.SP," where PID is a point type and SP is a parameter. A proto-permission can be applied to any object with the proper object type. The advantage is that the number of proto-permissions to manage is much smaller than the number of classical permissions that would have been needed. That is intended to satisfy Requirement 1. We note that this style of notation is used throughout the paper to represent records in the relational database, and should not be confused with functions.

Proto-permission groups. A proto-permission group (*ppg*) gathers proto-permissions, capturing typical access patterns associated with DCS job functions such as “operator,” “supervisor,” “engineer,” and “manager.” By grouping proto-permissions, a proto-permission group serves as a base template for a role.

Subjects. In our view, a “subject” exercises permissions, and so might be a human user, a particular device, or a software application. Subjects are assigned to roles, with different types of roles being defined for the different types of subjects.

User/application/device roles. A role *r* in DCS-RBAC is created with reference to a proto-permission group: *r(id; name; type; ppg:id)*. *r:type* designates a user, application, or device role. The specific objects for which a role has permissions are discovered through use of scopes. A scope, denoted by *sc*, associates roles with object groups; scopes are explained further later in this section. Conceptually, a role has a set of proto-permissions, and an associated super-set of objects that contains all objects to which those permissions can be applied. A match between a proto-permission’s object type and the type of a specific object gives the role access to that particular object, in accordance with the operation specified in the proto-permission. Together these features reduce the burden of managing a large number of permissions and allow us to leverage the common job functions and access patterns present in the DCS, satisfying Requirements 1 and 2. Furthermore, with proto-permission groups, role instantiation and management become easier and stay consistent with respect to the given job function: if a proto-permission is added (or removed) from the proto-permission group, all roles that derive from this group will automatically be updated. An example set of roles for a “Controller Zone A” follows:

r(1, Zone A Distillation Column Operator, user, operator ppg.id)

r(2, Zone A HMI, view only, app, hmi ppg.id)

r(3, Zone A Level 3 Control Station, device, lvl 3 control station ppg.id).

Role 1, for example, is the “Zone A Distillation Column Operator” user role that inherits all proto-permissions from the “operator” group (operator ppg). Likewise, Role 3 is a “Level 3 Control Station” device role that inherits proto-permissions from *lvl_3_control_station_ppg*. Application/device roles (roles 2 and 3) provide an intuitive, efficient, and fine-grained means to constrain the permissions that a user obtains when using a certain device or application. Without it, the role engineer would have to specify permissions for every possible combination of devices and applications for a given user role.

Assets and asset hierarchies. We borrow the term *asset* from DCS and use it synonymously with an *object* or *object group*. An asset has a *type* associated with it, indicating whether it is a “control” asset or a “system” asset*: a(id; name; type; treeId)*. For system assets, type may indicate more specific system assets like “console station,” “flex station,” “controller,” and so on. Systems of interest to us may have large number of assets, which are best handled through a well-structured hierarchy. Significant efficiencies in expressing permissions may be obtained by taking as the default assumption that permission granted to a role for an asset extends to all assets that are lower in the hierarchy. We express asset hierarchies using the *treeId* that shows where the asset is positioned in the asset hierarchy; this allows one to easily work out the hierarchical (parent-child) relationship between assets, and satisfies Requirement 3. Some example assets and asset hierarchies are shown below:

a(1, Control Assets, control, 1)

a(2, Zone A, control, 1.1)

a(3, Zone B, control, 1.2)

a(4, Cracker, control, 1.1.1)

a(5, Distillation, control, 1.1.2)

a(6, Control Loop 3, control, 1.1.2.1)

a(7, System Assets, system, 2)

a(8, Zone A, system, 2.1)

a(9, Stations, system, 2.1.2)

a(10, Zone A Console Station, system console station, 2.1.2.1)

a(11, Zone A Flex Station, system flex station, 2.1.2.2)

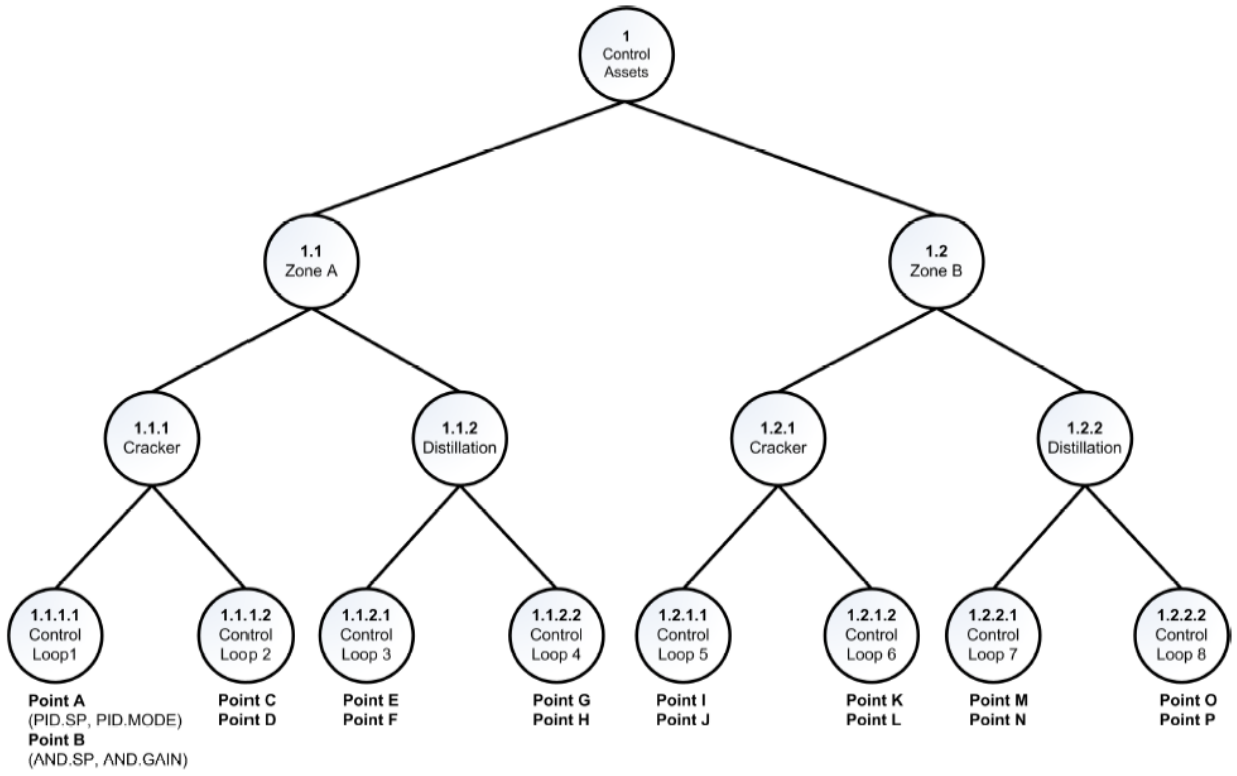


Figure 7 Control asset hierarchy example

Assets 1 to 6 are control assets, and assets 7 to 11 are system assets, where assets 1 and 7 represent the root assets in the two asset hierarchies. In the control asset hierarchy, “Zone A” (1.1) consists of “Cracker” (1.1.1) and “Distillation Column” (1.1.2), where Distillation Column is controlled by “Control Loop 3” (1.1.2.1). Figure 3 illustrates that control asset hierarchy. Similarly, in the system asset hierarchy, “Zone A” consists of “Consoles,” and “Stations.” Notice how the *a:type* for some system assets are more specific and indicate what the actual object types are (e.g., *system console station*). The reason is that the administrative proto-permissions are defined in terms of those system assets, which do not have any finer-grained objects (like points) associated with them. In comparison, a control asset, which is a logical grouping of control objects, never represents an accessible object or object type in itself; it may have (control) points assigned to it, though, and these points represent the objects on which the proto-permissions can be used.

Objects and proto-objects. In DCS, RBAC protects control points and their parameters. We use *pt* to denote a point: *pt(id; a:id; name; ptt:id)*. Each point references a type through *ptt:id*, and each type is associated with a set of parameters *par*. Each *ptt-par* combination, denoted by *pttPar*, represents a separate object; the reason is that a parameter, when associated with two different point (control algorithm) types, technically becomes two different parameter objects. *pttPar* is not a full object, however, since it is always bound to a point; such *ptt-par* combinations are referred to as ‘proto-objects’ in DCS-RBAC, and can be grouped together with point type. That is possible in DCS because a job function typically has the same access patterns for parameters regardless of the points to which they get assigned (see Requirement 2). For instance, an operator typically has a *write* access to the Set Point (SP) of a point type Proportional Integral Derivative (PID). That implies that for any point with point type PID, the operator can access its SP parameters. Grouping of proto-objects eases the object creation and management processes, requiring one to assign a small number of group identifiers (*ptt:id*) to the parent object (*pt*) to model a large number of associated attributes (*pt:pttPar*).

Exceptions. We support modifications to the default inheritance of proto-permissions by adding an exception *e* to the scope: *e(sc:id; a:treeId; ppg:id)*. A new proto-permission group is specified through *ppg:id*. One understands the complete proto-permission assignment as starting with the default, modified by changes expressed in the exceptions. This meets Requirement 3.

4.2.2 RBAC policy relational database schema

A relational database schema for DCS-RBAC is shown in Figure 8.

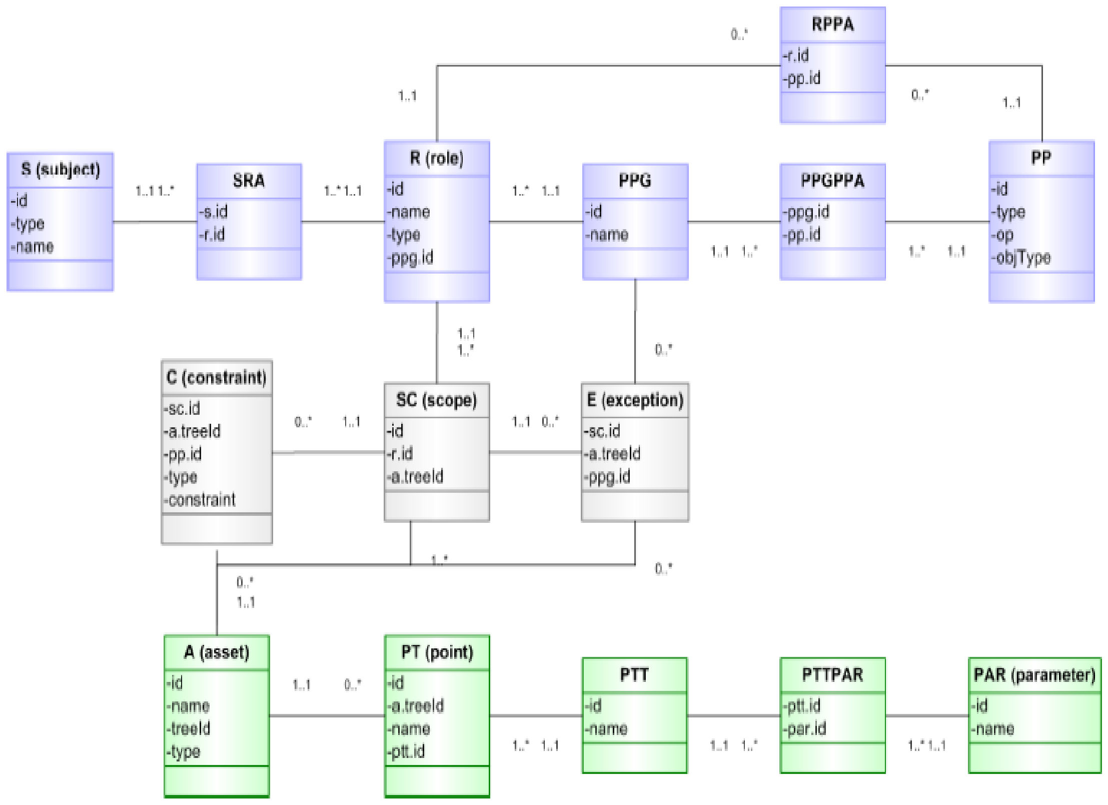


Figure 8 Relational database schema for DCS-RBAC

- Record **RPPA** links a role (with identity code *r:id*) directly with a proto-permission (with ID *pp:id*), capturing the *additional* proto-permissions (ones that are not part of the base proto-permission group) that the role can have. The **R** and **PP** records to which these IDs refer have the corresponding index values in their *id* fields, as is suggested by the links between records.
- Record **S** (subject) codes a user (a human, a device, or an application), giving it an identity (*id*), describes the type (human, device, or application), and an external identifier.
- Record **SRA** assigns a subject to a role, using *s:id* to index the subject record and *r:id* to index the role description.
- Record **R** (role) has an *id* that identifies it to the **RPPA** relation and the **S** (scope) relation. It has a unique textual *name* field, and references its base proto-permission group through the index *ppg:id*. It also has a *type* indicating whether it is a user, device, or application role.
- Record **PP** describes a proto-permission, giving it a unique index in field *id*, a type of proto-permission in field *type*, the operation allowed (described in field *op*), and the type of object that this proto-permission addresses.
- Record **PPG** gives a *name* and an *id* to a proto-permission group. Various assignments of proto-permissions to this group reference it through this *id* field.
- Record **PPGPPA** describes the assignment of the proto-permission (whose ID is *pp:id*) to a proto-permissions group, whose ID is *ppg:id*. This arrangement supports many-to-many linkages between proto-permissions and proto-permission groups.
- Record **SC** (scope) has a unique identifier *id*, identifies the role whose scope is through the role index *r:id*, and refers to a parent node in the asset hierarchy through code *a:treeId*.
- Record **C** (constraint) places restrictions on the use of proto-permissions. It points to a particular scope through field *sc:id*, and to the subtree of assets affected by the constraint through field *a:treeId*. The proto-permission is identified by index field *pp:id*; field *type* describes the constraint with labels such as “temporal” or “mode,” and field *constraint* gives a specific textual description (e.g., 7:00–18:00).
- Record **E** (exception) is used to allow modifications to a role’s base proto-permissions group. The scope being modified is identified through field *sc:id*; the asset subtree to which the exception is applied is identified through field a:treeId; and the group of proto-permissions used to replace all those formerly applied to that subtree is identified through field *ppg:id*. Multiple exception records may exist that point to the same scope record through *sc:id*.
- Record **A** (asset) has a unique identifier *id*, a textual *name* describing use (e.g., “Zone A”), a code *treeId* pointing to the asset’s position in the asset hierarchy, and a *type*.
- Record **PT** (point) has a unique identifier *id*, a position *a:treeId* pointing to its location in the asset hierarchy, a textual *name*, and an index to a point type. Although we do not discuss composite points (points with multiple point types) in this report, they can be modeled easily through a data structure that supports many-to-many relationships between point types and points.
- Record **PTT** describes a type for a point, with unique identifier *id*, and *name* of the point type in field name.

- Record **PTTPAR** is used to support many parameters a point type may have, and represents the proto-objects. It points to a specific parameter through index field *par:id*, and to a point type through index field *ptt:id*. A point, by referencing a point type, is also linked to the parameters.

A role's scope identifies a set of objects against which its proto-permission object types are matched to discover the objects (and operations on them) it can access. We explain later how this discovering process is managed by the policy decision point (PDP). The structure of the asset hierarchy means that it is simple to describe the scopes compactly. An asset and all of its descendents are described by the hierarchy coding of that asset; e.g., in our earlier examples, 1.1.1 is code for the control entity Cracker in Zone A. A role's scope may include a number of asset subtrees, and so is coded in the database as a list of scope records, one for each subtree.

The database schema supports discovery of all of a role's effective permissions to access assets, points, and parameters, and generation of "access vectors" used by the policy enforcement points (PEPs). The generation algorithm visits every control asset in the role's control scope, and, for each asset, fetches the associated points. For each point, it fetches the *pttPar* records (proto-objects) that have the matching point type. It also fetches every system asset in the role's administrative scope. All of these objects represent the *accessible objects* in the access vector. On top of those, the algorithm gets the role's scope records and the base proto-permission set (**PP** records) as well as the exceptions and new proto-permission sets associated with each exception. To grant access, the PEP searches the list of accessible objects and looks for a match of the object type with those given in the role's proto-permission sets. For a given proto-permission *pp*:

- if the PEP is examining a system asset *a*, and asset type *a:type* matches *pp:objType*, then the role may apply operator *pp:op* to *a*;
- if the PEP is examining point *pt*, and *pp:objType* is identically "point," then the role may apply operator *pp:op* to *pt*;
- if the PEP is examining *pttPar*, and *pttPar* matches *pp:objType*, then the role may apply operator *pp:op* to *pttPar* bound to the **PT** record that references the **PTT** record.

4.2.3 RBAC Policy Enforcement Framework

Figure 5 shows the placement of typical authorization framework components: the *centralized* RBAC Server is the policy decision point (PDP) that is responsible for managing the RBAC policy database and making policy decisions; each controller zone has its own policy enforcement point (PEP) that enforces the policies. In our architecture, the PEP mediates all user accesses to the data cache that caches field device (sensor) data. In typical policy-based authorization architectures, the PDP adjudicates access requests, based on policy, and the PEP enforces the decision. We eschew this approach in a DCS, as per-access network delays may threaten our response time requirements (see Requirement 5). In addition, the PDP would become a single point of failure, threatening safety (see Requirement 6).

Instead, our approach is to centrally, within the PDP, pre-compute access vectors that capture the RBAC policy, and push them out to each PEP. A per-role access vector contains (i) an encoding of the role's scope, (ii) all of the role's proto-permissions, and (iii) exceptions and constraints. Access vectors provide enough information for the PEP to correctly grant or deny a request. This pre-computation allows for efficient access enforcement within many different embedded devices that collectively constitute a distributed control system. The following steps describe how access control is enforced in our architecture:

(1) Offline, for every user role, application role, and device role, the PDP pre-computes per-role access vectors.

(2) Per-role access vectors, along with the list of subject-role assignments (SRA), are digitally signed and sent to the PEPs.

(3) On receiving an access request, a PEP uses the identity credentials of the user, application, and device and the SRA list to work out the three roles implied by the request. The PEP finds the appropriate access vector for each and for each role determines whether the request should be granted. Access is granted if and only if all three roles permit it.

(4) Whenever there is a policy change at the RBAC Server, the PDP computes the access vectors that are affected by the change, and pushes them out to the PEPs.

That approach is robust against PDP failures and network failures, as decisions are decentralized and made closer to the security objects. Note that while we place PEPs at the supervisory level (Level 2) in Figure 5, our framework allows us to push the PEPs down to actual controllers if needed to further decentralize access policy enforcement. However, controllers are embedded devices, and PEPs are required to store access vectors, so both memory and computing requirements might be a potential concern. As we will see, the list of proto-permissions in an access vector dominates the memory requirements. In this work, we focus on the case in which the PEPs are placed at a Data Cache at Level 2. Subsetting techniques are applied during the access vector generation process to reduce the size of access vectors in embedded controllers.

Access vector structure

We provide an example per-role access vector to show how it is structured and used to make authorization decisions. All the components of an example access vector are given below, represented using the same notations we used above for denoting database records.

role = r(1, 'Zone A Distillation Operator', user, operator ppg.id)

sc_ set(role) = {sc(1, role.id, 1.1.2), sc(2, role.id, 2.1.2)}

pp_ set(role) = {pp(1, point, view information, point), pp(2,parameter, write, PID.SP),
pp(3, parameter, view, PID.SP), pp(4, administrative, configure settings, system flex station)}

e_ set(sc(1, role.id, 1.1.2)) = {e(1, 1.1.2.1, view only.id)}
pp_ set(e(1, 1.1.2.1, view only.id)) ={pp(1, point, view information, point),
pp(2, parameter, view, PID.SP)}

object _set(role) = {pt(1, 1.1.2.1, 'Point-A', PID), pt(2,1.1.2, 'Point-B', PID), pttPar(PID, SP),
a(1, 'Zone A Flex Station', system flex station, 2.1.2.2)}

That vector was constructed for the "Zone A Distillation Operator" user role, consisting of a control scope *sc(1, role.id, 1.1.2)* and an administrative scope *sc(2, role.id, 2.1.2)*. The role has the proto-permission set of the operator group denoted by pp set(role). The control scope has an exception *e(1, 1.1.2.1, view_only.id)*, indicating that the new proto-permission set that can be used on all points and parameters that belong to asset 1.1.2.1 and its descendants is listed in *pp_ set(e(1;1.1.2.1; view_only:id))*.The set of objects that are accessible to the role are captured by *object_ set(role)*.

Now consider a user making a request to read parameter "Point-A.PID.SP." The PEP first checks that both "Point-A" and "PID.SP" are in *object_set(role)*. It then uses the *a:treeId* 1.1.2.1 of Point-A to figure out which set of proto-permissions can be used. Since *e(1, 1.1.2.1, view_only.id)* is the closest scope/exception to 1.1.2.1 in terms of the asset hierarchy, the PEP checks *pp_set(e(1, 1.1.2.1; view_only:id))* and grants the request, since this proto-permission set contains *pp(2, parameter, view, PID.SP)*. The object type checked here is PID.SP. However, a request to write to Point-A.PID.SP will be denied. A request to write to Point- B.PID.SP, on the other hand, will be granted, since here the proto-permission set that can be used is defined by *pp_set(role)*, which contains *pp(2, parameter, write, PID.SP)*. Similarly, a request to configure settings of "Zone A Flex Station" will be granted since *pp_set(role)* contains *pp(4, administrative, configure settings, system flex station)*. The object type checked here is *system_flex_station (a:type)*.

Note that this user role vector is just one vector being used in the policy enforcement process. The effective (final) access will be determined based on what the device and application roles allow.

Effective access vectors

Management and use of per-role access vectors is straightforward, but comes with its own costs: every run-time policy enforcement requires analysis of three different access vectors. Run-time responsiveness might be threatened, depending on the time cost of each analysis. It is possible to limit run-time analysis to one access vector, provided that more offline processing is done first. The idea is to visit every combination of user, device, and application role, and work out the *intersection* of their scopes, exceptions, and constraints. The access vector for the role-triple will hold a description of the intersecting scopes, and a list of the proto-permissions, exceptions, and constraints that are applicable to objects in that intersection. Loaded with such *effective access vectors*, a PEP handles a request by looking up the one vector created for the user/device/application triple making the request. The processing of the request is the same as for a per-role access vector.

An effective access vector is no larger (and is usually smaller) than a per-role access vector, owing to a smaller set of objects and proto-permissions in the intersection. On the other hand, there are potentially many more effective access vectors than there are per-role vectors. Which of the aforementioned two approaches is more efficient will depend very much on the number of possible role-triple combinations as well as the intersecting sets of scopes/exceptions and proto-permissions of effective access vectors. This represents a typical time vs. memory tradeoff.

The system implementation also used several access vector compression techniques:

- Object wild cards (e.g. read access to all objects)
- Permission wild cards (e.g. all access to specific object)

These techniques significantly compress access vector size in systems where read access is always granted for example and only write access is granted explicitly, which is typical in process control industry.

4.2.4 Performance Evaluation

To evaluate the performance of the proposed RBAC system against our processing time requirements, we constructed a prototype implementation for the RBAC Server, RBAC Proxy, and HMI using Microsoft SQL Server 2008 and C# (.NET) on Windows Server 2008. The RBAC Server, RBAC Proxy, and HMI were each deployed on a separate “Dell T310” server machine that is equipped with two quad-core Intel Xeon 3440 processors (2.53GHz) and 8GB memory. To model as closely as possible a real DCS architecture (see Figure 5 DCS overview with RBAC components), we used a Netgear router/firewall to connect the RBAC Proxy and HMI, and an unmanaged gigabit switch to connect the RBAC server to the router/- firewall. The router had OpenWrt installed and used default firewall settings. The RBAC Server prototype consists of a fully implemented RBAC policy database (see Figure 8) as well as a PDP engine for generating both the per-role and effective access vectors. The RBAC Proxy consists of a dummy data cache (containing point and parameter data) and a PEP engine that uses the access vectors to make authorization decisions. The HMI simply makes read and write requests to the Proxy.

Data size

According to DCS domain experts, a typical DCS has (approximately)

- 32 controllers;
- 2000 points per controller;
- 200 point types; 50 parameters per point type;
- at least four zones;
- at least 1000 assets.

We created an asset hierarchy in which the system has ten zones; each zone has ten large control entities (e.g., cracker); and each entity is controlled by nine control loops, for a total of 1010 assets in the hierarchy. The 32 × 2000 points in the system are distributed evenly among the assets, yielding about 64 points per asset. 200 ×50 proto-objects (PTTPAR records) were also created. There are 20,000 parameter proto-permissions on the proto-objects (200 point types × 50 parameters × 2 reads/writes). In addition, we created 12 point proto-permissions (e.g., view or configure algorithm information) and 20 administrative proto-permissions (e.g., view or configure settings).

To appreciate the expressive power of proto-objects (e.g., PID.SP) versus full objects (e.g., Point-A.PID.SP, Point-B.PID.SP), compare the 10,000 proto-objects with the 3,200,000 full objects the system would otherwise have to manage. Likewise, the expressive power of proto-permissions versus per-object permissions can be witnessed by comparing the 12 point proto-permissions with the 64,000 distinct point objects represented.

We grouped proto-permissions into six proto-permission groups, commonly used for creating six user roles, six device roles, and six application roles per zone.

Operation completion time and memory usage

To measure the operation completion time, we experimented with 20 different role combinations in each of four zones, and performed 100 read and 100 write requests on each access vector. These requests were generated randomly to test both "access-granted" and "access-denied" scenarios. The operation completion time for each request measured the time it took to submit the request through the HMI, authorize the request using the access vector, read the requested data from the data cache or write to the data cache entry, and send the response message back to the HMI. Table 1 shows the average operation completion time in milliseconds and the standard deviations for granted accesses, which involve reading from or writing to the data cache. It also shows the average time for denied accesses, which is shorter because there was no interaction with the cache. The statistics were collected using 2,000 read and 2,000 write requests per zone.

Significantly, all operation times were well under 50 milliseconds, which easily satisfies our response time requirement of less than a few hundred milliseconds. As anticipated, operational latencies for per-role access vectors were greater than those for the effective access vectors, but not significant enough to violate our response time requirement.

We also measured the time required to compute all 18 per-role vectors and 216 effective vectors for each of four zones. Statistics are shown in Table 2. Here we see a clear advantage to the per-role approach. While the performance difference is not prohibitive, especially given that computation is offline and infrequent, it does impact the responsiveness to a change in the RBAC policy.

Further, we measured the amount of memory needed by four of the zone PEPs to store access vectors, shown in Table 3. We see that the average size of effective vectors is much smaller than that of per-role vectors, but because there are more of them, in each zone the overall memory load on the PEP is smaller (by varying degrees) using the per-role vectors. While performance specifics will of course depend on the particular system (e.g., the number of needed roles), our study of a significantly large system shows that the proposed RBAC model and policy enforcement framework may likely meet the timing and memory requirements when the policies are enforced through centralized PEPs.

Comparing against full permissions

Finally, to demonstrate the size efficiency of our access vectors obtained through the use of proto-permissions, proto-objects, and object hierarchy, we expanded the per-role vectors into a format that contains *full permissions* – as opposed to proto-permissions and proto-objects – and measured their size. A full permission is a permission that one would normally see in a typical access control system, pairing an operation with an object instance. Without the proposed proto-permissions, proto-objects, and object hierarchy, an access vector would be generated with those full permissions and be larger in size.

As expected, the amount of total memory needed by the expanded per-role access vectors (compared with the original vectors) is much larger: at least 43% larger as shown in Tables 3 and 4. For Zone A, the difference is 12.8 MB, which is an increase of more than 80%; for the other three zones, the differences are about 5–6 MB which is an increase of more than 43%. A similar trend is seen for average access vector size. The difference is greater in Zone A because its roles have relatively large scopes and access to more objects. Likewise, as more roles are created with large scopes, the size difference between the two types of vectors will also become greater.

Table 5 shows that the operation times are slightly faster with the expanded vectors, owing to the simple (single) lookups on the full permissions for access decisions. We optimized the lookup times of the expanded vectors by modeling permissions with a *hashtable*. In comparison, the original vectors require a few extra steps (see Subsection 4.2.3) to figure out whether a user has access. Interestingly, the operation times for the expanded per-role vectors (which still require three access checks on user, device and application roles) are still slightly slower than those for the effective access vectors (see Table 1).

TABLE 1
Average operation completion time measured in milliseconds

	Per-roachle				*Effective*			
	Read (ms)	*Stdev*	*Write (ms)*	*Stdev*	*Read (ms)*	*Stdev*	*Write (ms)*	*Stdev*
Avg. time to grant	12.24	1.39	40.17	0.87	10.94	0.35	39.44	0.29
Avg. time to deny	2.98	0.64	3	0.57	2.06	0.17	1.98	0.12

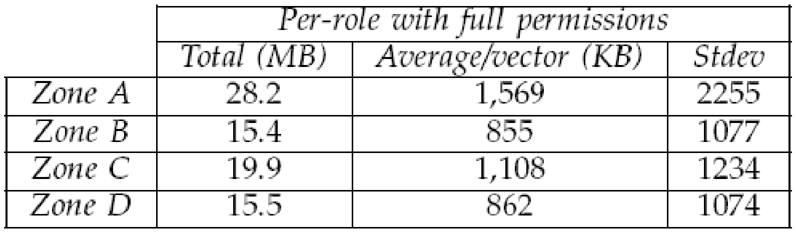
TABLE 2
Total access vector creation time in each of four zones

	Per-role			*Effective*		
	Total	*Average/vector (sec)*	*Stdev*	*Total*	*Average/vector (sec)*	*Stdev*
Zone A	7m 3s	24	18	14m 49s	4	7
Zone B	4m 23s	15	8	18m 49s	5	6
Zone C	6m 27s	22	15	31m 22s	9	10
Zone D	4m 33s	15	8	19m 7s	5	6

TABLE 3
Total access vector memory usage in each of four zones

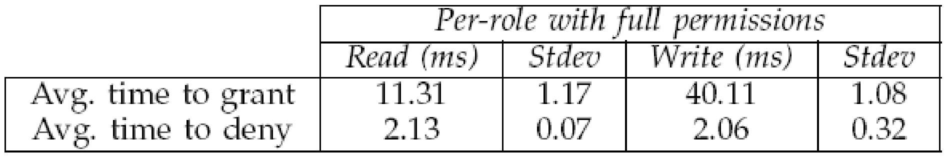
	Per-role			*Effective*		
	Total (MB)	*Average/vector (KB)*	*Stdev*	*Total (MB)*	*Average/vector (KB)*	*Stdev*
Zone A	15.4	854	756	16.2	75	205
Zone B	10.2	568	499	17	79	175
Zone C	13.9	772	680	20.2	94	280
Zone D	10.4	576	490	15.7	73	169

TABLE 4
Total access vector memory usage in each of four zones



	Per-role with full permissions		
	Total (MB)	*Average/vector (KB)*	*Stdev*
Zone A	28.2	1,569	2255
Zone B	15.4	855	1077
Zone C	19.9	1,108	1234
Zone D	15.5	862	1074

TABLE 5
Average operation completion time



	Per-role with full permissions			
	Read (ms)	*Stdev*	*Write (ms)*	*Stdev*
Avg. time to grant	11.31	1.17	40.11	1.08
Avg. time to deny	2.13	0.07	2.06	0.32

4.3 Formal Model

The previous subsection discussed a concrete DCS-RBAC model within a framework that meets the requirements outlined in section 4.1. That approach meets access control needs in DCS context, especially with respect to transitioning from current access control practices around Honeywell technologies. This section presents a formal model of RBAC that is informed by both the existing literature on RBAC models, and the unique requirements of RBAC in the process control space in general, and Honeywell's approach to that, in particular.

While subsection 4.2 describes elements of a potentially concrete implementation, what we develop here is at once more abstract, but also at times, more detailed. It explicitly captures the notion of scope of responsibility (and its various sublets) that is so important in the Honeywell context, but does not specify the mechanisms that determine the permissions allowed for a role to perform except in mathematical terms of relationships between sets. It brings to the foreground constraints on permissions related to context (e.g., system mode, time-of-day) through a novel dependency on ``environment state'', whereas constraints and constraint checking in 4.2 are more part of the implementation than part of the RBAC world-view.

The formal model has role-engineering firmly in its sights. It thoroughly works through means by which a role can be efficiently engineered through different inheritance mechanisms that suggest themselves by base role intent from the organization structure, from the relationship of objects within their asset hierarchies.

The formal model presents a unique approach to assigning permissions to roles, and users to roles through development of attribute-based policies. This formulation allows a role engineer to formally describe rules that guide and constrain the assignments, and then allow automation to create those assignments, at least as a starting point.

4.3.1 A Two-layered Framework of RBAC

The concept of role-based access control (RBAC) is simple, effectively models an organization's structure, and at a high level is easy to understand. However, the problem of defining roles and assigning permissions to roles is widely recognized as difficult, as is maintenance of RBAC as the organization changes both in structure, and with turnover in employment. In developing a formal RBAC model we have taken a two-pronged approach. First we describe RBAC at a level closely related to existing RBAC systems, at what we call the ``aboveground" layer. This description allows those already familiar with RBAC formalisms to readily understand the extensions we've made to support application in a process control context. As with other RBAC models, we describe this with mathematical notions involving sets and logical expressions based on set membership. A second layer, what we call the ``underground layer", describes how the abstractions of the upstairs layer might be realized in a way that captures the key distinctions in Honeywell's view of access control. The upstairs model aims to be descriptive of the structure of a broad set of RBAC systems; the downstairs model gives more details tailored towards specification of systems like Honeywell's. Taken together the aboveground and underground models completely describe an RBAC model suitable for process control systems; the relationship between the two is illustrated in Figure 9. It should be remembered though that the model is more general in several ways than is needed for Honeywell's implementation. It should also be remembered that the model concepts are more precise than needed for exposure to the user. We believe that the precision will be useful both for validating that a given RBAC implementation meets some higher level access control objectives, and will help guide our thinking as we consider problems related to role engineering. These two layers of the model give a sort of assembly language view of a program, while a user is accustomed to presentation at a higher and more concrete level.

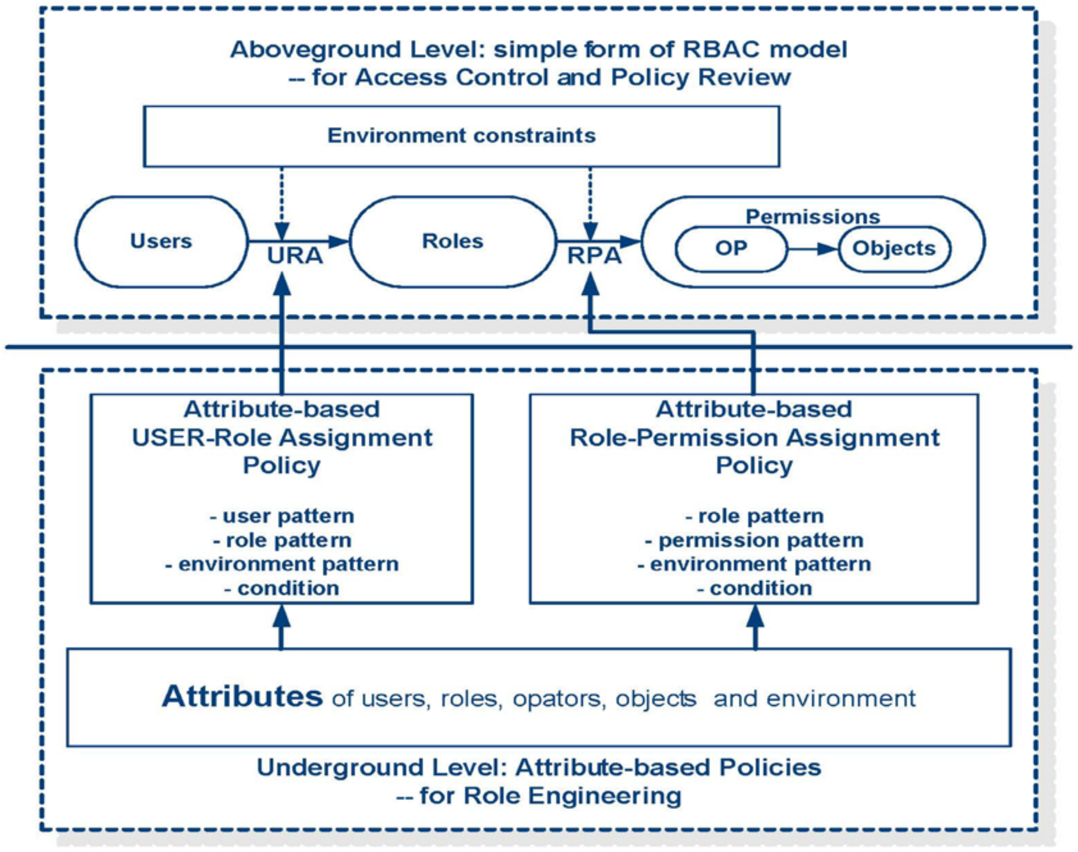


Figure 9 A two-layered framework integrating Attribute-based policies into RBAC

4.3.2 The Aboveground Level

In this section, we mainly use first order logic to make formal descriptions, and follow the convention that all unbound variables are universally quantified in the largest scope. The aboveground level is a simple and standard RBAC model, but extended with constraints on attributes of the "environment". We use the notion of environment to represent the context of a user's access, such as the time of access, the access device, the system's operational mode, and so forth.

The model is formally described as a tuple,

$$M = (U, R, P, O, OP, EP, URA^e, RPA^e),$$

where U is a set of *users*. A user could be either a human being or an autonomous software agent; R is a set of *roles*. A role reflects a job function, and is associated with a set of permissions; O is a set of *objects*. Objects are the resources protected by access control; OP is a set of *operator*. An operator represents a specific type of operations; P is a set of *permissions*. A permission is defined by a legitimate operation, which comprises an operator and an object; EP *is* a set of predefined *environment state patterns*. We use *environment state* to model the context of a user's access. Each environment state pattern (*environment pattern*, hereafter) defines a set of environment states; URA^e is the extended *user-role assignment* relation, which is basically a mapping from users to roles, associated with certain environment patterns; RPA^e is the extended *role-permission assignment* relation, which is basically a mapping from roles to permissions, also associated with certain environment patterns. All sets are finite. In the following, we give some further formal description of *environment*.

Environment

We represent the context of access as environment, and model an environment as a vector of environment attributes, each of which is represented by an environment variable (called an attribute name) associated with an attribute value in a domain. An environment is defined by n attributes, let $v_i \in D_i, i = 1, ..., n$, be the ith environment variable, where D_i is the domain of that environment variable; then a vector $(v_1, ..., v_n)$, in which all variables are instantiated is called an *environment state* (also denoted as s.) The set of all possible environment states is denoted by E. Choice of environment attributes (and hence environment state) is do- main dependent. Environment attributes, particularly the dynamic attributes, are gathered by an access control engine at runtime.

Example (environment state): Assume that the environment is defined by three attributes: mode, access location and access time; then mode = "normal" and access location = "station 1" and access time = "8:00AM Monday" is an environment state.

An *environment pattern*, denoted as e, is treated as an individual in domain EP, but is semantically defined by a first-order logical expression of assertions involving environmental attributes. An environment pattern defines a set of environment states, in which every environment state satisfies the environment pattern, i.e.

$$\{(v_1, ..., v_n) | e(v_1, ..., v_n)\}.$$

Hereafter, sometime we directly use e to denote the set of environment states defined by the environment pattern.

Examples (environment pattern): $Access\_location = station\_1$ and $access\_time \in [8:00, 22:00]$ is an environment pattern, which defines the set of all the environment states, having any mode, *access_location* at *station_ 1* , and *access_ time* between 8:00AM and 10:00PM.

We say that an environment state s matches environment pattern e, iff: $e(s)$ is true. An environment pattern can be empty, denoted by φ, which is most general; every state matches φ. We say that e_1 is subsumed by e_2, iff: the set of all environment states that match e_1 is a subset of the set that match e_2. That is, $subsume(e_2, e_1)$, iff: $\{s|e_1(s)\} \subseteq \{s|e_2(s)\}$. The relation is reflexive, transitive, and anti-symmetric.

User-role assignments

A particular user-role assignment associates a user, a role, and an environment pattern:

$$URA^e \subseteq U \times R \times EP, \qquad (1)$$

where EP is the set of all environment patterns that have been defined for the system of interest.

The semantics of a user-role assignment, $(u, r, e) \in URA^e$, is defined as:

$$\_match(s, e) \rightarrow has\_role(u, r), \qquad (2)$$

which states that if the real environment state s matches the given environment pattern e, then user u is assigned to role r. We assume that the RBAC engine could understand the semantics of each environment pattern as defined.

Basic RBAC models define user-role assignments simply as a mapping from users to roles,

$$URA^{e} \subseteq U \times R. \qquad (3)$$

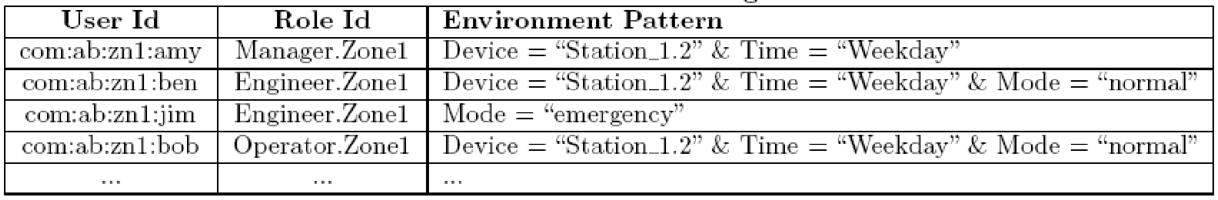
We have extended this notion with a dependency on the environment, as a means to integrate certain extended RBAC features of context and constraints. The environment pattern associated with a user-role assignment is the environment- dependent condition which is sufficient for the assignment. This feature can be regarded as constrained user-role assignment. If there are no constraints on user-role assignments, the associated environment patterns are simply empty, so the model becomes the common one.

In this model, the relation between URA^{e} and URA is:

$$(u, r) \in URA \leftrightarrow (\exists e, (u, r, e) \in URA^{e}). \qquad (4)$$

User-Role assignments may be expressed in tabular form. Table 1 shows an example, with an environment extension.

Table 1: Extended User-Role Assignment Table



User Id	Role Id	Environment Pattern
com:ab:zn1:amy	Manager.Zone1	Device = "Station_1.2" & Time = "Weekday"
com:ab:zn1:ben	Engineer.Zone1	Device = "Station_1.2" & Time = "Weekday" & Mode = "normal"
com:ab:zn1:jim	Engineer.Zone1	Mode = "emergency"
com:ab:zn1:bob	Operator.Zone1	Device = "Station_1.2" & Time = "Weekday" & Mode = "normal"
...	...	...

Role-permission assignments

A role-permission assignment associates a role, a permission, and an environment pattern. Thus the set of all such assignments is a subset

$$RPA^{e} \subseteq R \times P \times EP. \qquad (5)$$

The semantics of a role-permission assignment, $(r, p, e) \in RPA^{e}$, is defined as:

$$match(s, e) \rightarrow has\_permission(r, p), \qquad (6)$$

which states that if the real environment state s matches the pattern e, then permission p is assigned to role r.

Similar to user-role assignments, we've extended the common role-permission assignment with environment patterns.

The relation between RPA^{e} and RPA is:

$$(r, p) \in RPA \leftrightarrow (\exists e, (r, p, e) \in RPA^{e}). \qquad (7)$$

As with user-role assignments, the role-permission assignment may also be organized in tabular fashion.

4.3.3 The Underground Level

The underground level of RBAC model focuses on the security policies used to construct the aboveground level of RBAC model. Those security policies are a formalism of a role engineers' tacit knowledge about access control based on the attributes of users, roles, objects, and the environment, and the relation among them.

In the following, all users, roles, permissions, operators, and the security objects are treated as "*objects* " (in the sense of the object-oriented design) and each of which has certain attributes. *obj.attr*, or equivalently *attr* (*obj*), denotes the attribute *attr* of object *obj*.

The attributes needed in RBAC are typically domain dependent, and need to be customized for each specific target system. Some examples of attributes are as follows. The attributes of users may include "id", "department", "security clearance", "knowledge domain", "academic degree" or "professional certificate". A role may have attributes such as "name"; "type" reflecting job function types such as "manger", "engineer", and "operator"; "security level"; "professional requirements"; "direct superior roles" and "direct subordinate roles" (if role hierarchy is modeled). Objects may have attributes such as "id", "type", "security level", "state". Operators may have attributes like "name", and "type". The environment attributes may include "access time", "access location", "access application", "system mode", "target value", and so forth.

Role-Permission Assignment Policy

The role-permission assignment policy is a set of rules. Each rule has the following structure:

```
roleId{
    target {
-           role_pattern;
    -       permission_ pattern {
    -       operator_pattern;
  -         object _pattern;
    };
    environment_pattern;
    }
    condition;
    decision.
}
```

where, all of the *patterns* and the *condition* are FOL (First Order Logic) expressions; an *environment pattern*, as presented in 4.3.2, defines a set of environment states; similarly, a *role pattern* defines a set of roles by specifying their common attributes; a *permission pattern*, consisting of an operator pattern and one object pattern, defines a set of permissions by specifying their common features w.r.t. the attributes of the operator and the object in a permission; an *operator pattern* defines a set of operators (or operation types); each *object pattern* defines a set of objects; the *target*, which is, formally, the Cartesian product of the above sets of roles, permissions, and environment patterns, defines *the range of (role, permission, environment pattern) triples to which this rule applies* ; the *condition* is a logical expression defining a relation among the attributes of both the roles, the permissions (operators and objects), and the environment. This expression is the *condition under which a role-permission assignment can be made;* the *decision is* the role-permission assignment. This form of rules states that when the condition is true, a role covered by the role pattern can be assigned with a permission covered by the permission pattern in the specified environment pattern.

Let $pattern^R(r)$ denote a role pattern, referring the role as r, e.g. r.type="engineer"; $pattern^P(op, o)$ denote the general form of a permission pattern, referring to permissions of form $p(op, o)$. A permission pattern actually consists of zero or one operator pattern and zero or one object pattern; $pattern^E(\varepsilon)$ represents a predefined environment pattern ε; $condition(r, p(op, o), \varepsilon)$ denotes a logical expression, describing a relation among the attributes of role r, object o, and environment ε.

The semantics of a role-permission assignment rule is defined as follows.

$$(\forall r, op, o)(pattern^R(r) \wedge pattern^P(p(op, o)) \wedge pattern^E(\varepsilon) \wedge condition(r, p(op, o), \varepsilon) \rightarrow (r, p(op, o), \varepsilon) \in RPA^e) \quad (8)$$

which states that for any role r, satisfying the given role pat tern $pattern^R(r)$, for any permission $p(op, o)$, satisfying per- mission pattern $pattern^P(p(op, o))$, if $condition(r,p(op,o),\varepsilon)$ is true, then role r is assigned with permission $p(op,o)$ in environment pattern ε.

A pattern can be empty, φ. An empty pattern defines the most general pattern, which every element in a domain matches. For example, if role pattern is empty, then the defined role-permission assignment rule is applied to all of the roles in R. Therefore, if the target of a rule is empty, then the rule is most general and applicable to all combinations of role, permission, and environment; on the other hand, a rule is most specific, (i.e. only applicable to a single specific combination,) if the target of the rule is defined most specifically, i.e. the role pattern is exactly defined as a specific role instance, the permission pattern is exactly defined as a specific permission/operation, and the environment pattern is also most specific. Generally, the target of a rule defines a specified range of (role, permission, environment) triples to which the rule applies.

In this framework, a role could be defined based on a role template. Different from the concept of "role" in RBAC, a role template is associated with a set of permission patterns rather than permissions. The idea is developed to address scaling issues that arise in the DCS domain.

For simplicity, we only use positive rules. If (r, p, e) is in RPA^e, then role r is assigned with permission p in environment e; if not, by the close world assumption, we conclude that role r does not have permission p in environment e.

In role engineering, the defined role-permission assignment policy (rule set) is applied to all possible combinations of (role instance, permission, environment pattern), and if a combination can be inferred by any rule, then it is included in the RPA^e on the aboveground level of RBAC model.

User-Role Assignment Policy

User-role assignment is highly dependent on business rules and constraints. In our view, the task of assigning users to roles can be approached like the role-permission assignment problem, in terms of policies that enforce those rules and constraints.   Such policies would be formulated in terms of user and role attributes, and would be crafted to enforce things like Separation of Duty. However, unlike role- permission assignment, user assignment may have to balance competing or conflicting policy rules against each other. Correspondingly a complete policy oriented formulation will need to specify how to combine rules and arrive at a final assignment decision. In what we present below, an attribute based user-role assignment policy is used only to identify potential assignments. A rule-combining algorithm is used for making the final assignments.

Similar to the role-permission assignment rule, the user-role assignment rule consists of:

```
rule id {
    target {
      user_pattern;
      role_pattern;
      environment_pattern;
    }
    condition;
    decision.
}
```

where, similarly, an user/role/environment pattern defines a set of users/roles/environment states by specifying the common attributes; all of the *patterns* and the *condition* are expressions in first order logic; differently, the *decision* of a rule is *to mark an (user, role, environemt_pattern) triple as a potential assignment*.

Let $pattern^U(u)$ denote a user pattern, referring to user u; $pattern^R(r)$ denote a role pattern, referring to role r; $pattern^E(\varepsilon)$ denote the specification of an environment pattern; $condition(u, r, \varepsilon)$ denote a logical expression, describing a relation among the attributes of user u, role r, and environment pattern ε.

The semantics of a user-role assignment rule is defined as follows.

$$(\forall u, r)(pattern\ U(u) \wedge pattern\ R(r) \wedge pattern\ E(\varepsilon) \wedge condition(u, r, \varepsilon) \rightarrow (u, r, \varepsilon) \in temp\ \ URAe), \qquad (9)$$

which states that for any user u satisfying $pattern^U(u)$, any role r satisfying $pattern^R(r)$, if the condition that states a relation among the attributes of u, r, and ε is true, then $(u, r, \varepsilon) \in temp\_URA^e$, which means that according to this rule, u can be assigned to r in environment ε.

Example:

rule:{

```
    target:{
        role_pattern(r): r.type  = "chemical engineer";
        environment_pattern (e):{
            Device = "Station_1.2"
            and  Time  = "Weekday"
            and  Mode = "Normal" }
    }
    condition:{
        knowledge_match(u,r);
        security_req_match (u,r);
        u.base_plant = "Houston";
    }
    decision:  add  (u,r,e)  in $URA^e$.
}
```

In this rule, knowledge match(*u*, *r*) is a function which returns true if the professional knowledge of user *u* matches the knowledge requirements of role *r*; security_req_match(*u*, *r*) is a function which returns true if the user meets the security requirements of role *r*. A necessary condition for considering assigning a user *u* to role *r* is that the user work at the Houston plant (where presumably one finds Station 1.2.)

The rule has no filter on the users it considers. The rule applies to roles with job type attribute “chemical engineer”. Now suppose that there is a role “Engineer.Zone.1.2” designed to work in Zone 1, on Station 1.2, on weekdays; this role has job type attribute chemical engineer, requires professional knowledge in Chemical Engineering, and at least a Bachelor’s degree in that field. Going through employees one finds John, who works at the Houston plant; John has a Bachelor degree on Chemical Engineering; and suppose that John meets the security requirements for role “Engineer.Zone.1.2”. The patterns for user and role cover these instances, the condition of the rule is true, and so the assignment of John to that role will be marked as possible. Of course, there may be other roles that John is suitable for, and there may be other employees suitable for role “Engineer.Zone.1.2”, so a later step is needed to select among all assignments marked as possible, by considering some constraints such as separation of duty.

The following is the pseudo-code of rule combining algorithm.

```
rule_combining(temp_$URA^e$) {
    for (i =  0;  i <  constraints.length(); i++){
        if ( ! satisfy(temp_$URA^e$,  constraints[i])){
            add  i in  conflictList;
        }
    }
```

```
        if ( conflictList.length() == 0)  { $URA^e$ =  $temp\_URA^e$;
        }  else  {
                if (constraintConflictResolution($temp\_URA^e$, conflictList, $temp2\_URA^e$))
                        $URA^e$  =  $temp2\_URA^e$;
                else
                        notify(temp_URAe,  conflictList);
        }
}
```

Note that the constraint conflict resolution algorithm is dependent on constraints, which in turn are domain-dependent, so it is difficult to give a general algorithm for constraint conflict resolution. If conflict cannot be resolved by algorithm, the algorithm will inform RBAC administrator, and the conflict resolution needs to be performed manually (or with tools we have not identified) by the role-engineer.

4.4 Summary

Starting from today's practice of DCS in Honeywell, particular the practice of Experion system, we developed a highly usable RBAC model for DCS (DCS-RBAC) that facilitates smooth migration from Experion systems to RBAC and ease the management of roles, permissions, and security objects. This approach meets the requirements of RBAC in the control system context.

We also developed a rigorous formal RBAC model for DCS, which has two layers - the aboveground and underground levels. This formalism keeps the simplicity of RBAC in the aboveground level, and hides the complexity of the knowledge used to create RBAC model in the underground level. The underground level uses attribute-based policies to manage URAs (user to role assignments) and RPAs (role to permission assignments).

5 Appendix A - Terms, Acronyms and Abbreviations

Below are the terms, acronyms, and abbreviations used within this document.

AES - Advanced Encryption Standard

API – Application programming interface

BITW – Bump In The Wire

CA – Certificate Authority

Certificate - X.509 v3 public key certificate used to authenticate a device or function.

CRL – Certificate Revocation List

CSR – Certificate Signing Request

DCS - Distributed Control System

ESP - Encapsulating Security Payload

GMAC - Galois Message Authentication Code

HMAC - hash-based message authentication code

HMI - Human Machine Interfaces

HPS - Honeywell Process Solutions

HUG – Honeywell user group

IKE - Internet key exchange

INL - Idaho National Labs

IPsec - Internet Protocol Security

IT – Information Technology

ITI - Information Trust Institute

LDAP – Lightweight Directory Access Protocol

MITM - man in the middle

PA – Policy Agent

PAA - Policy audit and analysis

PBKDF2 - Password-Based Key Derivation Function 2

PDP - policy decision point

PEP - policy enforcement point

PID - Proportional-Integral-Derivate

PKI - public-key infrastructure

PLC - programmable logic controller

NFR - Non Functional Requirement

RFC – Request for comment

SA – Security Association

SCADA - Supervisory Control and Data Acquisition

SCEP - Simple Certificate Enrollment Protocol

SP – Set Point

SPI - Security Parameters Index

SRS - Software Requirements Specification

TLS - Transport Layer Security
UIUC - University of Illinois at Urbana Champaign
VPN - virtual private network