



Pacific Northwest
NATIONAL LABORATORY

*Proudly Operated by **Battelle** Since 1965*

Transaction-Based Building Controls Framework, Volume 2: Platform Descriptive Model and Requirements

BA Akyol
JN Haack
BJ Carpenter
S Katipamula
RG Lutes
G Hernandez

July 2015

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.** Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY

operated by

BATTELLE

for the

UNITED STATES DEPARTMENT OF ENERGY

under Contract DE-AC05-76RL01830



This document was printed on recycled paper.

(9/2003)

Transaction-Based Building Controls Framework, Volume 2: Platform Descriptive Model and Requirements

BA Akyol
JN Haack
BJ Carpenter
S Katipamula
RG Lutes
G Hernandez

July 2015

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

Executive Summary

Given the distributed nature of the power grid—with millions of components operating independently and without context of their environments—centralized control is difficult and cost prohibitive to implement at scale. Transaction-based Building Controls (TBC) offer a control systems centric platform that provides an agent-based execution environment that meets the growing requirements for security, resource utilization, and reliability. This report outlines the requirements for a platform to meet the definition, requirements, and characteristics of TBC and describes an illustrative/exemplary implementation TBC.

VOLTTRON™ is an open-source distributed control and sensing platform developed by Pacific Northwest National Laboratory for the Department of Energy for use by the Office of Energy Efficiency and Renewable Energy to support Transactional Energy activities. VOLTTRON™ is designed to be an overarching integration platform to bring together vendors, users, and developers and enable rapid application development and testing. This document summarizes the VOLTTRON™ components, platform details, applications, and security features specific to building to grid. VOLTTRON™ is a minimally viable TBC solution supported by the U.S. Government.

The solution- and design-level requirements for a TBC platform are the primary focus of this document. Goal- and domain-level requirements are featured in a companion report¹. This report is not meant to be an exhaustive design specification but instead discusses the components and modules that satisfy a TBC platform. The goal of this report is to provide an explanation of what needs to be built and to encourage multiple, interoperable TBC platform implementations in order to build up an ecosystem of platforms working together to achieve the goals of Transactional Energy.

¹ Available at http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-23302.pdf

Acknowledgements

The authors would like to express their gratitude to Joe Hagerman, Senior Technical Advisor at the Building Technologies Office, for his guidance while drafting this report.

Contents

| | | |
|-------|---|------|
| 1.0 | Introduction | 9 |
| 1.1 | TBC Platform Requirements | 10 |
| 1.2 | U.S. Department of Energy Role..... | 11 |
| 1.3 | Report Organization..... | 1.1 |
| 2.0 | Generic TBC Platform Requirements..... | 2.2 |
| 2.1 | TBC Platform Solution-Level Requirements | 2.2 |
| 2.2 | TBC Platform Design-Level Requirements | 2.5 |
| 2.2.1 | Hardware Requirements..... | 2.5 |
| 2.2.2 | Operating System Requirements..... | 2.5 |
| 2.2.3 | Security Requirements | 2.6 |
| 2.2.4 | Communications and Networking Requirements for TBC Platforms | 2.9 |
| 2.2.5 | Diagnostic and Manageability Requirements | 2.9 |
| 2.2.6 | Information Sharing Requirements | 2.9 |
| 2.2.7 | TBC Platform Agent Execution Environment Requirements | 2.10 |
| 3.0 | VOLTTRON™: An Example TBC Platform..... | 3.1 |
| 3.1 | Security Overview..... | 3.2 |
| 3.2 | Components of the VOLTTRON™ Platform | 3.3 |
| 3.3 | Component Details..... | 3.5 |
| 3.4 | VOLTTRON™ Platform Details | 3.17 |
| 3.4.1 | VOLTTRON™ Platform Services..... | 3.18 |
| 3.4.2 | Agent Lifecycle..... | 3.18 |
| 3.5 | Applications Built Using the VOLTTRON™ Platform..... | 3.23 |
| 3.5.1 | Example Energy Efficiency Agent/Application Deployment on TBC | 3.24 |
| 3.5.2 | AFDD Utilization of Platform Services | 3.25 |
| 3.5.3 | Grid Service Agent/Application..... | 3.26 |
| 3.6 | Security Features of VOLTTRON™ | 3.28 |
| 3.6.1 | Vulnerabilities Associated with Communications between VOLTTRON™ and Other Services | 3.1 |
| 3.6.2 | Vulnerabilities Associated with Communications between Multiple VOLTTRON™ Instances | 3.2 |
| 3.6.3 | Vulnerabilities Associated with Agent Multi-Tenancy inside a VOLTTRON™ Platform..... | 3.4 |
| 3.6.4 | Vulnerabilities Associated with Communicating with and Controlling Devices | 3.5 |
| 3.6.5 | Vulnerabilities Associated with the Underlying Operating System and File System..... | 3.6 |
| 3.6.6 | Vulnerabilities associated with user administration of the platform, user interfaces, etc. | 3.8 |
| 3.6.7 | Summary of VOLTTRON™ Security Features | 3.9 |

Appendix A - An Example Best Practice for Securing Building Control Networks A.1
Appendix B - Linux Platform Hardening for VOLTTRON™B.1

Figures

| | |
|---|------|
| 1.1. Levels of Increasing Specificity of Requirements | 10 |
| 3.1. Components of the VOLTTRON™ platform. Cybersecurity is a cross-cutting feature of the platform covered in detail in section 3.6..... | 3.1 |
| 3.2. Schematic of VOLTTRON™ Internal and External Components of the platform | 3.3 |
| 3.3. An agent publishes a message matching the subscription of another agent..... | 3.6 |
| 3.4. Diagram of an agent receiving data from and issuing commands to a device | 3.6 |
| 3.5. All actors involved in a VOLTTRON™ deployment communicate via the IEB | 3.7 |
| 3.6. Example of command-line output..... | 3.8 |
| 3.7. Administrator provisions remote VOLTTRON™ instances by sending prepared agents (represented by green dots) to them..... | 3.9 |
| 3.8. A running agent moves to a remote platform by sending a request to the Mobility Module... | 3.9 |
| 3.9. Communication between VOLTTRON™ platforms via the MultiNode service | 3.10 |
| 3.10. The Actuator coordinates agent control requests with driver agents and the scheduler | 3.11 |
| 3.11. VOLTTRON™ drivers are the interface between devices and the platform. They can talk directly to devices or through a controller device..... | 3.12 |
| 3.12. Management Console exposed as a web interface for use by administrators | 3.13 |
| 3.13. Historian agents store data from the message bus in a local cache until successfully writing to external storage | 3.15 |
| 3.14. Agent requesting control for a device | 3.16 |
| 3.15. The Weather Service acts as a proxy allowing agents to interact with a web service via the message bus | 3.17 |
| 3.16. Workflow for Agent Execution..... | 3.19 |
| 3.17. Workflow for agent movement | 3.20 |
| 3.18. Workflow for Agent Upgrade | 3.21 |
| 3.19. Components of the Agent Transport Payload | 3.22 |
| 3.20. An Example Deployment of VOLTTRON™ in an Enterprise..... | 3.24 |
| 3.21. Schematic of VOLTTRON™ Internal and External Components of the platform | 3.29 |

1.0 Introduction

The distributed nature of the evolving power grid, such as responsive loads, solar and wind generation, and automation in the distribution system present a complex environment not easily controlled in a centralized manner. Buildings include many site located, complex systems that interact with each other as well as the power grid. In an environment where so many components operate independently and without complete knowledge of their environments, centralized control is very difficult to implement – costly to “get it right” with high integration costs, vendor specific in terms of insuring complete operation of all device features, and often requires costly best available equipment (not commodity class equipment).

Fundamentally, distributive software agents are designed to accomplish tasks on their own based on information from their environment and other agents, thus the potential for local decision making and cooperation makes agents a natural fit for controlling complex distributed systems. If only equipment could take advantage of these agents. Deploying agents into a control systems including buildings-grid integration requires software to meet strict requirements for security, resource utilization, and reliability. These requirements, as observed by the U.S. Government in early 2010 after widespread deployment of smart grid technologies supported by American Recovery and Reinvestment Act investments, can delay or prevent the transition from research to deployment and fundamentally limit the development of complex control solutions.

Transaction-based Building Controls (TBC) offer a control systems platform that provides an agent execution environment fulfilling security requirements, empowering agent developers to avoid solving these problems independently on their own and speed up the time to deployment and implementation.

This report outlines the requirements for a TBC platform – addressing the needs of complex controls and describes an illustrative/exemplary implementation leveraging VOLTTRON™. This implementation captures the requirements in various components, which are described in more detail in section 3.3. This report is not meant to be an exhaustive design specification but instead discusses the components and modules that make up a minimally viable (and the illustrative example of a) TBC platform. The goal of this document is to provide a thorough discussion and explanation of TBC needs to encourage multiple, interoperable TBC platform implementations in order to build up an ecosystem of platforms working together to achieve the goals of Transactional Energy.

This document outlines the requirements and describes an illustrative/exemplary implementation. This implementation captures the requirements in various components, which are described in more detail in section 3.3.

1.1 TBC Platform Requirements



Figure 1.1. Levels of Increasing Specificity of Requirements

Figure 1.1 describes four levels of TBC platform requirements: Goal, Domain, Solution, and Design. Both solution- and design-level requirements are featured in this report; goal- and domain-level requirements are featured in the companion report².

These four levels of requirements are defined as follows:

- **Goal-Level Requirements:** These are the requirements of transaction-based energy services aimed at 1) improving the operational efficiency of buildings and the overall stability and control of the power grid and 2) serving other high-level operational objectives of the numerous stakeholders involved. This level of requirements motivates holding workshops, developing frameworks, and providing incentives for a smart building and a smart grid based on transactional energy and energy-related services. The goal-level requirements for the implementation of transactional energy services are covered in the companion report (Somasundaram et al. 2014³).
- **Domain-Level Requirements:** Requirements at this level describe the range and types of transactional services needed to serve the higher-level goals, who needs to participate in them, and how they interact with each other. The set of transactional energy use cases described in the companion report provide descriptive scenarios that are essential at this domain level. The domain-level requirements for the implementation of transactional energy services are discussed in the companion report.
- **Solution-Level Requirements:** These requirements necessarily assume a general form for the technology system that meets the higher-level requirements as well as some concept of the actual technology that would provide that solution in terms of the operational environment and conditions under which it would be installed and used, who would administer it, and such. Solution-level requirements are discussed in this report.

² Accessible at http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-23302.pdf

³ Somasundaram S, RG Pratt, BA Akyol, N Fernandez, NAF Foster, S Katipamula, ET Mayhorn, A Somani, AC Steckley, and ZT Taylor. 2014. [Transaction-Based Building Controls Framework, Volume 1: Reference Guide. PNNL-23302](#), Pacific Northwest National Laboratory, Richland, WA

- **Design-Level Requirements:** These requirements assume both the general form of the solution and the actual technology assumptions for choices of hardware and software, communication protocols used, and economic constraints to produce, install, and operate. Design-level requirements provide the details necessary to design a particular implementation of the technology solution and are discussed in this report.

These different levels of requirements generally build on each other from the broader goal level down to the very specific design level. Each higher level of requirements should be well defined to provide the context for defining requirements at the next lower level.

Requirements at the higher levels tend to be expressed as qualitative behaviors, or what software practitioners call non-functional requirements. At the lower levels, requirements necessarily take on more functional specificity. Design-level requirements, for example, are generally comprised of functional requirements described with enough detail to allow an actual test with specific conditions and results to be defined and performed on the final technology solution so as to gauge whether the requirement has indeed been met.

1.2 U.S. Department of Energy Role

The U.S. Department of Energy (DOE) aims to enable and promote the development of efficient, secure, and reliable transaction-based energy services, markets, and operations that integrate energy supply, demand, and related services to promote a diverse, reliable, cost effective, and sustainable domestic energy economy. DOE believes that the future energy model will include an open, interoperable, transaction-based system that facilitates physical transactions of energy, energy-related services and rights, and the financial settlements associated with these transactions.

As many buildings lack infrastructure to allow either one-way or two-way communications with external entities, software agents can connect devices and accomplish tasks on their own based on information from their environment and other agents to support transaction services. TBC is a supervisory control systems platform that provides an agent execution environment that fulfills controls, management and security requirements, significantly raising buildings' energy efficiency. Yet to enable TBC, several technological advances in the area of cost-effective building automation systems are needed. The information listed in this report intends to support interested parties with a detailed listing of the functionality of the software agent, its components, requirements, and the architecture of it so that they can enhance control technology to more efficiently manage building related loads. The detailed descriptions of specific use cases described in this report are meant to facilitate understanding of how TBCs in buildings work, what benefits they entail, and how industry, the research community and vendors can realize some of the new technologies described. DOE hopes to solicit comments and feedback on the listed approach, allowing stakeholders to participate in developing and improving TBCs, so that buildings, as well the power system and society as a whole, can benefit from its promise of reduced energy consumption and emissions.

1.3 Report Organization

Although the language of “must,” “shall,” or “needs to” is often used in describing requirements, this language at the higher levels often implies necessity, whereas at the lower levels it is often specifying a contingent choice. In other words, a particular requirement that says “It must ...” at a goal or domain level is likely describing something that is necessary for the system to be valid at all, whereas the same language for a requirement at the solution or design level may be defining a particular functionality or attribute of the solution that could have been chosen differently. A formal set of functional requirements at the design level are the purview of the design process for a specific vendor implementation of a TBC platform and are outside the scope of this report.

The remainder of this report is organized as follows:

- The TBC Platform requirements are described in Section 2.0 and Section 3.0 uses an exemplary TBC to orient the user and to provide a concrete footing based on the TBC platform requirements.
 - Section 3.1 presents an overview the exemplary TBC platform.
 - Section 3.2 gives an overview of the components.
 - Section 3.3 describes the solution- and design-level requirements in detail.
 - Section 3.4 describes in detail an exemplary platform that can be used to meet these requirements.
 - Section 3.5 presents use cases that describe how a particular instantiation of a TBC platform was used to implement building HVAC system diagnostics and identify faults and to perform demand response.
 - Section 3.6 presents the security features of the exemplary platform.
- The appendices provide an example deployment and recommendations for hardening the host platform for a TBC.

2.0 Generic TBC Platform Requirements

Engineering for any product or tool starts with a requirements analysis phase. The previous sections described VOLTTRON™ as an exemplary TBC platform to orient the reader. This section describes solution- and design-level requirements for a TBC platform. The goal of this section is to define the requirements with *sufficient* detail so that any vendor or organization can build a TBC that is able to implement the applications and use cases discussed in the companion volume of this report (Somasundaram et al. 2014¹). A formal set of functional requirements at the design level are the purview of the design process for a specific vendor implementation of a TBC platform and are outside the scope of this report.

2.1 TBC Platform Solution-Level Requirements

Solution-level requirements drive engineering design choices. Solution-level requirements assume not only a general form for the technology system that meets the higher-level requirements but also some concept of the actual technology that would provide that solution in terms of the operational environment and conditions under which it would be installed and used.

From a buildings' and facilities' technology perspective, commercial and residential buildings are complex environments. The ownership may change every few years. The deployed technology within the buildings may be as old as few decades (if not more). A building owner or operator may need to integrate management of several types of buildings into a single system and interface that with the local energy distribution utilities. To manage such a diverse environment, TBC platforms must be open, flexible, modular, and scalable as summarized below:

- A TBC platform must support common computer processing unit (CPU) and hardware architectures and be able to run on small form-factor boards such as ARM based Raspberry PI², Panda and Beagle Bone Black³ boards, and other platforms such as Intel NUC⁴.
- TBC platforms must be designed to scale up and down in performance and functionality in order to maintain functionality on devices and equipment of differing computational power and placed in different physical locations to support transactive energy at many layers in a building.
- TBC platforms must be open-source to encourage community participation, allow vendors to contribute to the system, and encourage innovation. Although the TBC platforms may be built on open-source solutions, vendors can add proprietary components.
- TBC platforms must provide highly flexible application programming interfaces to shield application developers from the complexities of building energy-efficiency services, building control applications, and the power grid.

¹ Somasundaram S, RG Pratt, BA Akyol, N Fernandez, NAF Foster, S Katipamula, ET Mayhorn, A Somani, AC Steckley, and ZT Taylor. 2014. [Transaction-Based Building Controls Framework, Volume 1: Reference Guide, PNNL-23302](#), Pacific Northwest National Laboratory, Richland, WA

² <http://www.raspberrypi.org/>

³ <http://beagleboard.org/BLACK>

⁴ <http://www.intel.com/content/www/us/en/nuc/overview.html>

- TBC platforms must run on modern operating systems and not be tied to a specific operating system. Any features tied to the specifics of an operating system must be re-implementable in other operating systems or be optional.
- TBC platforms must be built using modern programming techniques and provide an easy-to-use development environment for developers that includes system utilities, building application service providers, and example applications. This ensures maximum participation in transactive energy framework.
- TBC platforms must provide application programming interfaces to Cloud-computing platforms such as Amazon AWS, Microsoft Azure, Google, etc. Built-in Cloud-computing support will allow system integrators to build flexible and scalable management solutions that can use high-performance computing or extensible storage as necessary.
- A TBC platform must be able to communicate with other TBC platforms using the framework's standard communications protocol. See section 2.2.4 for a detailed description of the communication requirements. The remote platform interface implements this protocol so that the TBC platform can interact with other nodes in the transaction-based energy services network regardless of which technologies or architectures are implemented by the remote platforms.
- Interoperability is a primary requirement of a TBC platform. Each node within a TBC network must use a standard protocol for communicating with other nodes within the network in order to effectively transact “exchangeable” services. The nodes with which a TBC platform interacts may have entirely different implementations and may use different architectures and choices of technology solutions. However, they must be able to communicate a set of standard messages with each other over a common communication protocol. The protocol must define both the syntax and the semantics for a set of functional services sufficient to perform all of the transactional energy-service communications between the two nodal platforms.
- TBC platforms must provide built-in facilities to support transactive energy applications including but not limited to:
 - a built-in historian;
 - mathematical, statistical, and computational libraries for facilitating and validating transactions;
 - a transaction settlement mechanism using cryptographic means for correct attribution and non-repudiation;
 - a cryptographically, authenticated transaction journal for settlement validation;
 - weather and historical climate data interface to allow modeling of climate conditions to construct forecast patterns for consumption;
 - synchronized time service to support both transactions and to enable correct handling of cryptographic identities;
 - data collection export service for validating operations, debugging, and maintenance; and
 - persistent storage for storing application data.
- TBC platforms must not force developers to use a single language; they must support transactive energy applications in many modern computer programming languages. While the main utilities and base classes may be written in the main language of the TBC platforms, other common programming

languages shall be supported and can easily be extended. TBC platforms must provide a mechanism for acting as a proxy for legacy devices that cannot be enhanced to support transactive energy in order to preserve investment in devices and equipment and help transition older buildings to the new world of transactive energy.

- TBC platforms must support secure automated application download and provisioning from the network so that new transactive energy applications can be deployed quickly and flexibly and be kept up to date.
- TBC platforms must support secure, automated software updates from the network. The TBC platform must preserve existing configuration and application settings during and after a software upgrade.
- TBC platforms must allow the user or the administrator to recover the platform to a known valid state. The TBC platform must allow configuration export and import from the network to allow an administrator to restore the platform to a known state.
- TBC platforms must provide a means for maintaining transactional integrity across the time span of the network or communications outages.
- TBC platforms must be able to store the information for equipment configurations, schedules, and user preferences through any network or power outages.
- Many of the transactions involved in the transactional energy services require measurement and verification of energy performance of the end-use devices. TBC platforms must ensure that the parties are meeting the obligations in an agreed-upon transaction in order to complete the transaction and adhere to the terms of a contract or other agreement governing the transaction.
- TBC platforms must support real-time and non-real time applications.
- TBC platforms must provide mechanisms to scale upwards and manage thousands of TBC platform devices, perform updates, and change configuration.
- TBC platforms must be able to run multiple applications simultaneously and securely to maximize the value from integrating transactive energy into a building.
- TBC platforms must provide an interface connecting the platform's logical processing components to the actual hardware devices that the platform needs to monitor and control. The device interface is representative of a direct physical connection for controlling and monitoring device hardware, such as thermostats, HVAC equipment, lights, sensors, electric vehicles, PV generators, etc.
- TBC platforms must prioritize and/or provide exclusive (write/control) access to devices being controlled by the platform.
- TBC platforms must include a built-in scheduler for applications to reserve time slots for control of devices, buildings, or other resources.
- TBC platforms must provide an automatic data gathering service to collect device performance and historical data and make it available to all applications via a messaging bus supporting multi-point communications. If an application only needs access to collected data, they do not need to reserve exclusive access in order to perform their functions.

- TBC platforms must provide a capability discovery mechanism in order for platforms and agents to discover each other.
- TBC platforms must be built using widely accepted, industry-standard cyber security controls and monitoring technologies. TBC platforms are expected to implement cyber security mechanisms to assure integrity, availability, and confidentiality as described in publications such as NIST SP800-53 and NIST SP800-82 as well as other widely accepted industry standards.

2.2 TBC Platform Design-Level Requirements

Design-level requirements assume not only the general form of the solution but actual technology assumptions for choices of hardware and software, communication protocols used, and economic constraints to produce, install, and operate. Design-level requirements provide the details necessary to design a particular implementation of the technology solution and are discussed in this section. The design-level requirements are in addition to solution-level requirements defined in section 2.1.

2.2.1 Hardware Requirements

TBC platform hardware requirements can be separated into CPU, memory, storage, and inputs/outputs. For CPUs, 32-bit addressing and a memory management unit that supports virtual memory, paging, and process memory segmentation are highly desirable. For example, memory segmentation enables different agents running on the system to be isolated from each other and improve the cyber security of the platform.

The memory (RAM) located on a device used in a building control system may be less than what is now common in consumer applications. Typically in control systems, flash memory is used and the operating system must be aware of wear associated with flash memory when information is written to the same sectors repeatedly. For the purpose of this report, the platform will target devices with 1024MB or more memory while paying attention to detail to ensure that the platform can be sized down to lower memory sizes. The platform will also target devices with at least 8GB non-volatile storage.

A control system may have many inputs and outputs. TBC platform discussed in this report has no specific requirements on inputs and outputs. However, since the TBC platforms are assumed to be networked, they have at least one wired or wireless network interface.

2.2.2 Operating System Requirements

The capabilities developed for the TBC platform will be *operating system agnostic*. Given the memory, storage, and CPU capabilities of target devices, the chosen operating system must be capable of running on devices with a memory of 1024MB and reserve no less than 512MB of memory for transaction-based controls agent framework. It is recommended to use POSIX APIs wherever possible. POSIX APIs are supported in many operating systems including Linux.

2.2.3 Security Requirements

A TBC platform must provide security against unauthorized access to data and control functionality. The TBC platform must also isolate applications running on the platform from each other (if needed) and enforce resource utilization limits on the applications to assure stability. The TBC platform must use standardized security mechanisms including encryption, authentication and authorization mechanisms.

Communications with other platforms must include identification, authentication, and authorization functions to ensure that only legitimate transactions are performed. Access to the system through local user interfaces should also include these security measures.

The security requirements may differ between different transactional service applications depending on the particular aspects of the service and the participants involved. Each transactional service application must specify its security characteristics and requirements. These security characteristics include:

- Level of security and data encryption required for communication transmissions and for data storage.
- Identification requirements of parties participating in the transactions. For example, attributes such as non-repudiation, cryptographic mechanisms that are being used to establish identities (e.g., X.509 certificates), and the establishment of trust roots may need to be specified.
- Security requirements/features for the communication patterns being used by the application. For example, are there any firewalls or is network address translation expected to be in the communication paths? Does it require push or pull for messaging? Is the application event driven? Does the application require cross-domain transactions? How are cross-domain transactions handled? Is there a third-party broker involved? How do we accommodate different cyber security authorization policies? Is there a trusted intermediary (e.g., escrow)?
- Transactions validation requirements. Is there an instantaneous enforcement required to ensure against incorrect or malicious actions? Is there a reconciliation mechanism to ensure compliant behavior? If the transactive energy messages are modified during transit, how is the system to ensure that the modifications are not erroneous or malicious?

When considering the security properties of confidentiality, integrity, and availability, the control systems community typically takes a much different approach than would typically be taken by the government, military, banking industry, etc. that places the highest priority on confidentiality and the lowest on availability. The viability of the control systems depends on placing the highest priority on availability and the lowest on confidentiality. Between these two extremes is the security property of integrity (source integrity/identity and data integrity) that contributes to reliable operation by preventing system compromise and by rapidly detecting any system compromise that does occur. The Authentication and Authorization component of the framework directly addresses integrity.

R. Shirey¹ defines four broad classes of threats – disclosure, deception, disruption, and usurpation of some part of a system. Disclosure is a violation of confidentiality. Deception and disruption are generally

¹ R. Shirey, *Security Architecture for Internet Protocols: A Guide for Protocol Designs and Standards*, Internet Draft: draft-irtf-psrg-secarch-sect1-00.txt (Nov. 1994).

countered by integrity services. Disruption is also countered by availability mechanisms.¹ Availability, integrity, and confidentiality requirements are discussed below.

2.2.3.1 Availability Requirements

- TBC platforms must be able to detect resource exhaustion (either malicious or due to defective software) and take corrective action. A specific case of resource exhaustion is when a TBC platform is bombarded with high rates of incoming network traffic.
- A specific case of resource exhaustion occurs when TBC platforms face high data rates of incoming traffic causing malfunction or a crash. Therefore, TBC platforms must be able to handle incoming network traffic at the full data rate supported by its interfaces. Especially for platforms with limited processing power; this requirement may be met by means of hardware support or by means of a separate security device. This requirement is necessary to prevent brute-force denial-of-service (DoS) attacks. For battery-operated platforms, platforms must be able to detect behavior that causes battery exhaustion and take corrective action.
- TBC platforms must be resilient against hardware failures.
- TBC platforms must be able to continue operation when network connections are temporarily disrupted.
- TBC platforms must detect and terminate an application that is not behaving according to its application/platform contract.
- TBC platforms must implement a “liveness” check for each application hosted on the platform in order to detect and terminate malfunctioning applications. Applications must be written to respond correctly with this check or they may be stopped prematurely.
- TBC platforms must support sending security related logs to a remote security event information management system.
- Cryptographic measures must not prevent the platform from meeting its expected service levels.

2.2.3.2 Integrity Requirements

- Device hardware must provide interfaces to the TBC platform to accommodate physical tamper detection and warning systems and respond to an event generated by the tamper detection circuitry.
- Integrity service design must not reduce availability or performance below service-level agreements set by the administrators of an organization.
- Agents running on the TBC platform must be authenticated and authorized at every step of the agent lifecycle including agent instantiation and movement.
- Authorization must be configurable via policy statements.
- TBC platform must be able to detect if code or data have been tampered. Although there are increased risks without prevention, prevention is not a requirement due to the impact on control system availability.

¹ Matt Bishop, Introduction to Computer Security, Addison-Wesley, 2005.

- TBC platforms must validate both the platform and the applications as part of the boot process.
- TBC platforms must periodically validate the software to detect tampered applications if the software framework resides in modifiable storage.
- TBC platforms must include the means to lock certain memory locations as read-only once the boot process is completed. This may include all cryptographic and authentication libraries.
- TBC platforms must work with openly available key management systems.
- TBC platforms must be able to detect bad or poorly written software agents. This tracking may be implemented by means of a reputation-based tracking system.
- TBC platform must periodically validate itself to detect tampered executables if the software framework resides in modifiable storage.
- TBC platform must offer memory protection to hosted agents and platform services. (This feature may be provided by the operating system.)
- TBC platform may lock certain memory locations as read-only once the boot process is completed. This may include all cryptographic and authentication libraries.
- TBC platforms must provide a mechanism to automatically check and receive updates for both security and maintenance from a known, secure resource on the network.
- TBC platforms must be able to meet authentication, authorization, and accounting requirements as discussed in NIST SP800-82/NIST SP800-53 as well as requirements for communication protocols security. Additional cyber security requirements may also be found at “Guidelines for Smart Grid Cyber Security” published by NIST IR7628¹.
- TBC platforms must include a mechanism for tracking the reputation of applications. Applications that consistently exceed their resource contract, crash, or fail to respond to platform directives will be prevented from running on the platform.
- TBC platform shall notify any administrators of all agent-related security failures.

2.2.3.3 Confidentiality Requirements

Confidentiality concerns arise in regard to business interests and personal privacy and also secrecy if the agent is, for example, monitoring for illegal usage. Confidentiality requirements include:

- The developer must be able to recommend confidentiality for the agent.
- The administrator must be able to configure the agent for confidentiality.
- If the agent is configured for confidentiality, the source must encrypt the payload for the destination device prior to moving the agent.
- The source device must be held accountable for sending a confidential agent without encrypting the payload.
- TBC platforms must support cryptographic algorithms as recommended in FIPS 140-2 or later.

¹ http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf

2.2.4 Communications and Networking Requirements for TBC Platforms

There are many types of protocols and communications infrastructure in use in residential and commercial buildings. To facilitate successful integration of transactive energy into existing buildings, TBC platforms must include built-in support for a broad range of devices and protocols and must be easily extensible to support other protocols.

TBC platforms must provide built-in and flexible support for modern control system protocols, and new protocols should be easily and quickly added to TBC platforms. Example protocols can be Modbus, BACnet, LON, DNP3, SEP 2.0, OpenADR (Open Automated Demand Response), etc. The key requirement is for the platform to make it easy to integrate new device control systems and data gathering protocols.

TBC platforms must communicate over many types of communication networks including IEEE 802.11, IEEE 802.15.4 and IEEE 802.3 networks. The TBC platforms can utilize a gateway or a protocol converter to communicate over legacy serial links.

TBC platforms must include supporting both IPv4 and IPv6 communications and can use both TCP (transmission control protocol) and UDP (user datagram protocol) over IPv4 and IPv6.

The number of devices that can be controlled or monitored by a TBC platform must be limited by only the hardware and communications capabilities on which the platform is deployed. The TBC platform itself should be able to scale up or down based on the underlying platform hardware capabilities without inherent limits in the software.

2.2.5 Diagnostic and Manageability Requirements

One of the advantages of using agents in the framework is the ease of diagnostics and management. Agents can be dispatched to collect data as they move through the system and then carry that data back with them to a management station. The TBC platform must be able to manage the activities of all hosted agents. The management may include stopping, restarting or *refusing to execute based on configuration policies* an agent deployed in the system. The TBC platform needs to provide a logging service based on the ubiquitous SYSLOG format for reporting diagnostics including security events.

2.2.6 Information Sharing Requirements

To be able to share information is a key tenet of any distributed system. The agents in the power system are capable of accomplishing tasks that would require large amounts of communication if one tries to implement the same capability using a centralized system. However, to prevent an information overload and keep information as secure as possible, how much information is shared and where it is shared needs to be carefully addressed. The agent-based software framework that DOE envisions must be able to accommodate many-to-many secure and reliable information exchange.

2.2.7 TBC Platform Agent Execution Environment Requirements

Software agents are entities that can either be stationary or roam between sensors in the system. Agents are task driven and can be used to accomplish a task that involves many sensors. The following are the requirements for the TBC platform agent execution environment:

- Agents run on multiple CPU architectures as long as agent resource requirements are met. The platform must provide the agents with a runtime environment independent of the underlying CPU architecture.
- Agent execution environment must be able to determine whether it has the capabilities to support a particular agent. In essence, the platform and the agent enter into a binding contract when a platform hosts an agent.
- Agent execution environment must be able to determine if the agent requesting "hosting" is authorized to execute on that platform.
- Agent execution environment must be able to determine if the agent code has been tampered with.
- Agent execution environment must allow agents to store information on-board.
- Agent execution environment must provide access to information collection and dissemination services for the agent.
- Agent execution environment must be able to guarantee (stochastically) a requested slice of its resources such as memory, storage, and processor cycles to an agent.
- Agent execution environment must be able to detect and terminate an agent that is not behaving according to its agent/platform contract.
- Agent execution environment must be able to collect and transmit an inventory of agents hosted by the platform.
- Agent execution environment must implement a "liveness" check for each agent hosted on the platform in order to detect and terminate malfunctioning agents.
- Agent execution environment must provide a protected environment for each agent. This could include memory and storage protection.
- If the platform restarts, existing agents will be restarted as well. There is no guarantee of a complete state restore. Agents should write sufficient data to the local store to ensure that they can recover.

3.0 VOLTTRON™: An Example TBC Platform

VOLTTRON™ is an open-source distributed control and sensing platform developed by Pacific Northwest National Laboratory (PNNL) for DOE for use by the Office of Energy Efficiency and Renewable Energy to support Transactional Energy activities. The key principle adopted by VOLTTRON™ developers is flexibility and openness. VOLTTRON™ is not tied to a single application (e.g., demand response) or a single protocol (e.g., OpenADR, SEP2.0). The platform is designed to be an overarching integration platform to bring together vendors, users, and developers and enable rapid application development and testing. This document summarizes the VOLTTRON™ components, platform details, applications, and security features specific to building to grid. The platform is a minimally viable TBC solution supported by the U.S. Government. The major modules/services of VOLTTRON™ are shown in Figure 3.1.

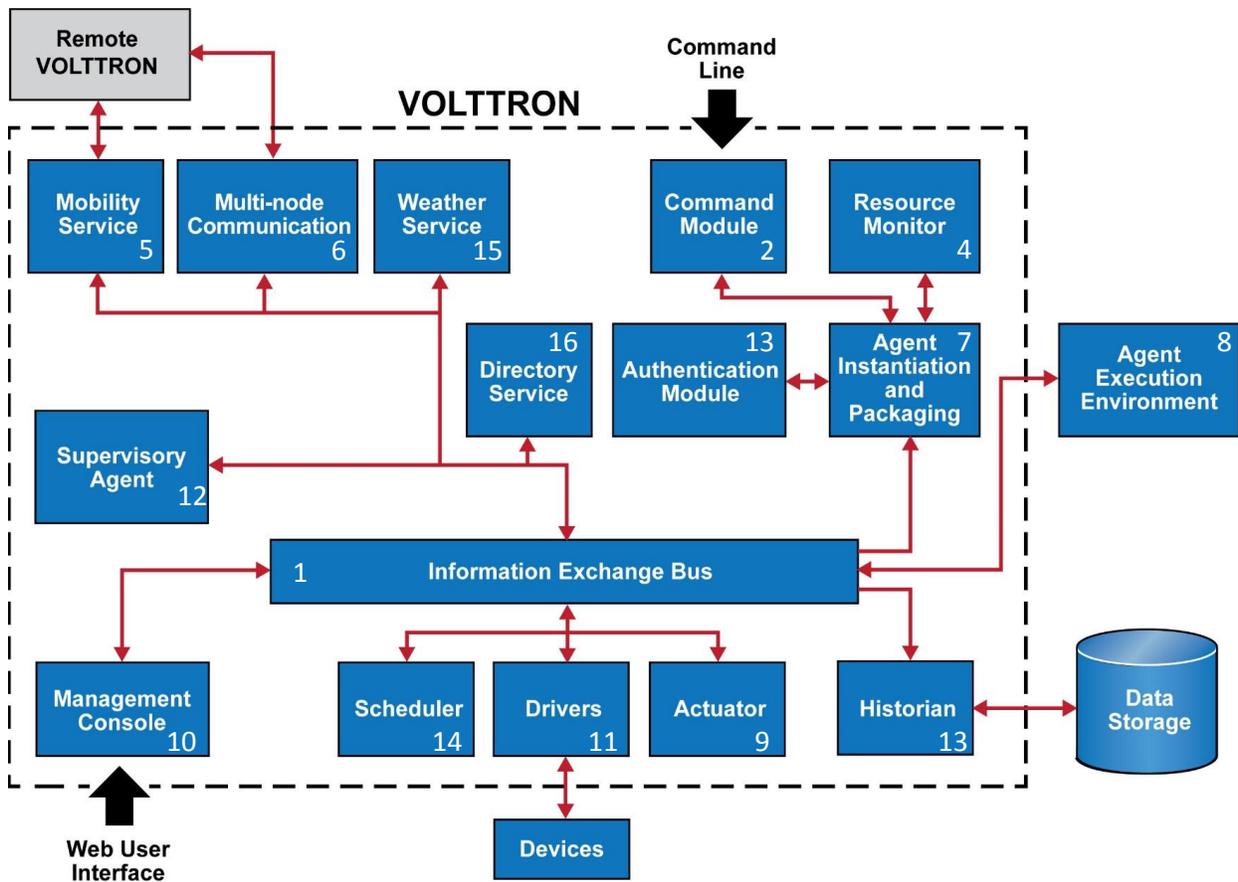


Figure 3.1. Components of the VOLTTRON™ platform. Cybersecurity is a cross-cutting feature of the platform covered in detail in section 3.6.

3.1 Security Overview

VOLTTRON™ enables rapid authoring and secure deployment of autonomous software agents for distributed sensing and controls. It is designed to be as secure as possible to meet desired security objectives. Therefore, the platform uses a threat model approach for determining threats and vulnerabilities of the software to reasonably reduce the attack surface and/or harm endured after a compromise. The first and second releases of VOLTTRON™ (1.0 and 2.0) focused on:

- Protecting the integrity of agent programming through cryptographic means
- Protecting agents from using excessive system resources to prevent platform instability
- Protecting agent configuration (and work orders) from manipulation
- Securing communications between VOLTTRON™ platforms and external data sources
- Securing communications between platform instances, including the transfer of agents.

This work was completed with the release of VOLTTRON™ 2.0 and assumed that agent code will be well vetted for correctness and assessed for malicious intent. Currently, this strategy works because of the limited number of VOLTTRON™ deployments and existing agents are being developed by a small group of trusted developers. As VOLTTRON™ usage increases, the number of agent developers is expected to increase, driving the need for additional security features. Future versions of the platform will include additions to improve agent trust and integrity, including agent message authentication and encryption and full agent “containerization” or “sandboxing.”

Figure 3.2 shows two VOLTTRON™ systems communicating with each other and with external sources. The system on the left side is expanded to show internal components of the platform and communication with an external historian and a Cloud-based service. While VOLTTRON™ addresses threats associated with the platform and its interfaces, equal thought needs to be given to the security of the underlying operating system. For more details on the security features, see section 3.6 of this document.

More detailed information about VOLTTRON™ is available at at <https://github.com/VOLTTRON/volttron/wiki>.

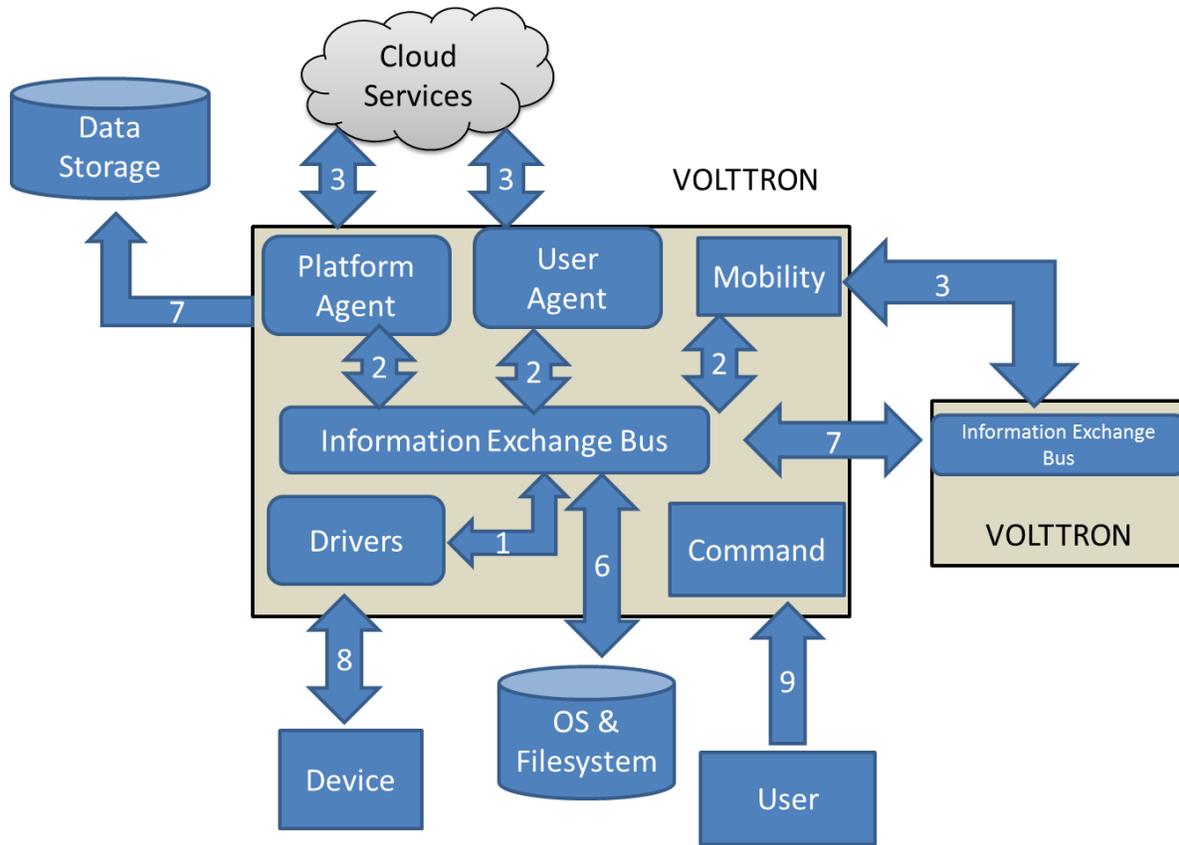


Figure 3.2. Schematic of VOLTRON™ Internal and External Components of the platform

3.2 Components of the VOLTRON™ Platform

The major components of the VOLTRON™ platform are summarized below with a reference to the TBC component they represent. The platform components are discussed in greater detail in section 3.3.

- Information Exchange Bus (IEB) (also referred to as the message bus by VOLTRON™ users): The IEB is one of the most important components of VOLTRON™. Agents running on the platform can communicate with each other and with platform services over a common publish/subscribe based message bus. IEB simplifies interfacing between agents written in different languages and with devices controlled by the platform. IEB acts as the switchboard between agents on the platform and platform services.
- Command Module: The platform has a command framework that allows agents to be loaded and unloaded dynamically during execution. This feature allows agents to be updated without rebooting the platform.
- Authorization Module: Before agents are allowed to run on the platform, they must be verified by this module. The contents of the agent package are checked and the signatures are validated.
- Resource Manager (RM): Determines if sufficient resources exist to fulfill an agent's execution contract and monitors existing agents to ensure they do not exceed its terms.

- **Mobility Service (MS):** Allows an administrator to send agents to other platforms and also allows agents to request to be moved. Works with the Agent Instantiation and Packaging (AIP) to verify agent packages then instantiate and launch them.
- **MultiNode Communication:** Allows agents on a platform to publish and subscribe to the IEB on remote platforms.
- **Agent Instantiation and Packaging (AIP):** Coordinates the verification of agent payloads through the Authorization Module and determination of sufficient resources through the RM.
- **Agent Execution Environment (AEE):** Agents execute in an environment managed by the platform to provide proper access controls and resource management. The platform can support multiple environments that allow agents to be written in any language.
- **Actuator:** Serves as an intermediary for agents issuing commands to devices controlled by the platform.
- **Drivers:** Allows the platform to interact with devices and building controllers by providing an abstraction layer that handles device communication. Drivers communicate using the appropriate protocol (MODBUS, BACnet, etc.) with controlled devices and keep the protocol specific details isolated from the rest of the platform.
- **Management Console:** Gives administrators a view into the status of platforms under their control.
- **Supervisory (Management) Agent:** Monitors the state of the platform and resets devices to known good states in case of a failure.
- **Historian:** Subscribes to readings from devices and messages from agents then stores them for later retrieval either locally or remotely.
- **Scheduler:** Allows agents to reserve access to devices to prevent conflicting commands coming from multiple agents.
- **Weather Service:** A Cloud agent that provides weather information to other agents by interfacing with a web service.
- **Directory Service:** A service that lists available agents and their capabilities.

Another component of the platform are the agents deployed within it. Agents can have different roles based on whether they provide services to other agents or perform specific tasks as part of an application built on VOLTRON™. These roles can be broadly classified into the following groups:

- **Platform Agents:** Agents that are part of the platform and provide a service to other agents. Examples are agents that interact with devices to publish readings and handle control signals from other agents.
- **Cloud Agents:** These agents represent a remote application that needs access to the messages and data on the platform. This agent can subscribe to topics of interest to the remote application and can also allow it to publish data to the platform.
- **Control Agents:** These agents control the devices of interest and interact with other resources to achieve a particular goal.

3.3 Component Details

VOLTTRON™ comprises several components that provide the functionality needed to transport and receive agents, verify their authenticity, allow them to communicate within and between platforms, and ensure that they do not exceed the host's resources. The following sections describe these components in more detail.

3.3.1.1 Information Exchange Bus (annotated as component “1” in the Figure 3.1)

The IEB is the heart of the VOLTTRON™ software. It allows agents and services to exchange data using ZeroMQ¹ (0MQ). ZeroMQ is a free software widely used by National Aeronautics and Space Administration (NASA), Cisco, etc. to provide scalable, reliable, and fast communication. ZeroMQ provides a unified socket-based API for sending and receiving messages across multiple protocols such as IPC, multicast, TCP, etc. With client libraries for multiple programming languages, ZeroMQ allows VOLTTRON™ to support agents written in any language.

ZeroMQ supports many topologies, of which VOLTTRON™ is using publish/subscribe. The platform's agents and services publish multipart messages consisting of the topic, message headers, and message content. By using this convention, VOLTTRON™ allows entities to subscribe to messages of interest by matching the topic and headers. All agents and services can publish and subscribe to topics on the message bus. This provides a single and uniform interface that abstracts the details of devices and agents from each other. At the most basic level, agents and components running in the platform produce and consume messages and/or events. The details of how agents produce events and how they process received events are left up to the agents. For agents written in Python, VOLTTRON™ provides a base agent class as well as utility functions that greatly simplify publishing and subscribing to messages.

Agents and services running on the platform can communicate with each other over a common message bus. This feature simplifies interfacing between agents written in different languages and with devices controlled by the platform. IEB provides a messaging bus as a scalable multi-language solution for inter-agent communication both within a platform and in-between platforms. Agents post to topics and other agents can subscribe to their events if they are interested. This capability simplifies agents as they receive messages and produce their own messages. Agents, who need to work together, can establish their own topics for communication. Utilities in the platform allow agents to have methods triggered by a message coming through on a subscribed topic that meets certain requirements.

Agents can cooperate over the message bus (Figure 3.3) by sending and receiving data over a topic. For example, an agent could use data from devices to come up with a predicted value it then publishes which is used by other applications to decide when to take a control action.

An example of agents interacting with devices via the message bus can be found in the Figure 3.4. The driver for an HVAC unit publishes readings from its data points to the message bus (1). The agent is subscribed to the “OutdoorAirTemp” topic and receives data on its subscription (2). The agent, which already has a reservation on the device (see Figure 3.4, publishes a command to the Actuator topic (3). The Actuator agent receives that message on its subscription (4), verifies the reservation, then issues the command to the HVAC unit (5).

¹ <http://zeromq.org/>

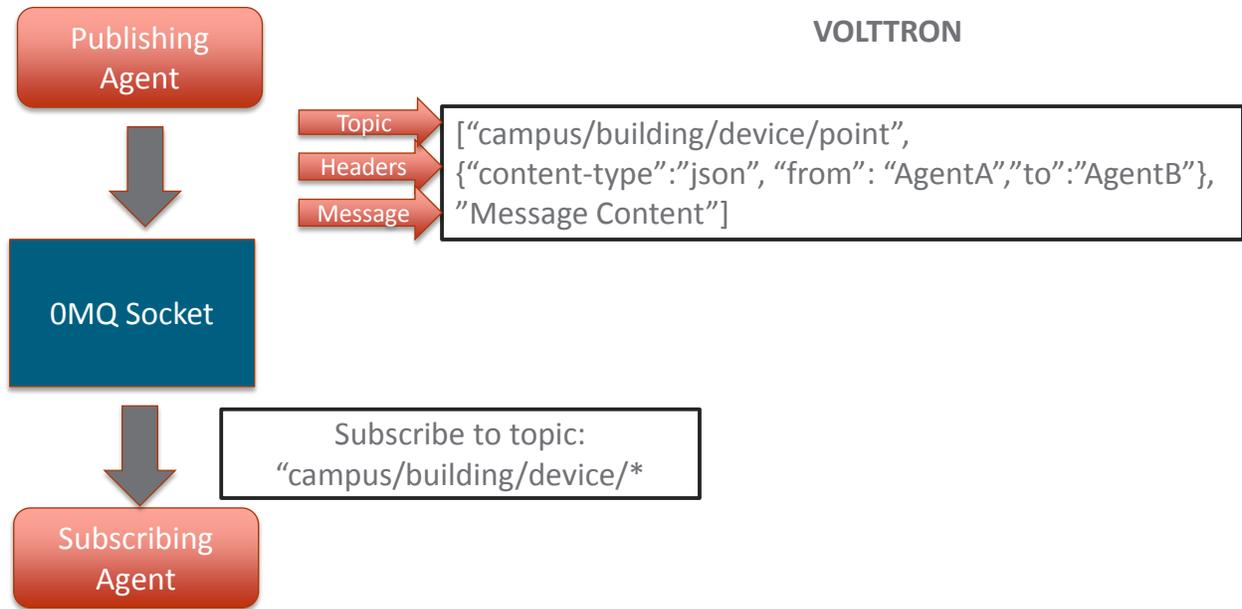


Figure 3.3. An agent publishes a message matching the subscription of another agent

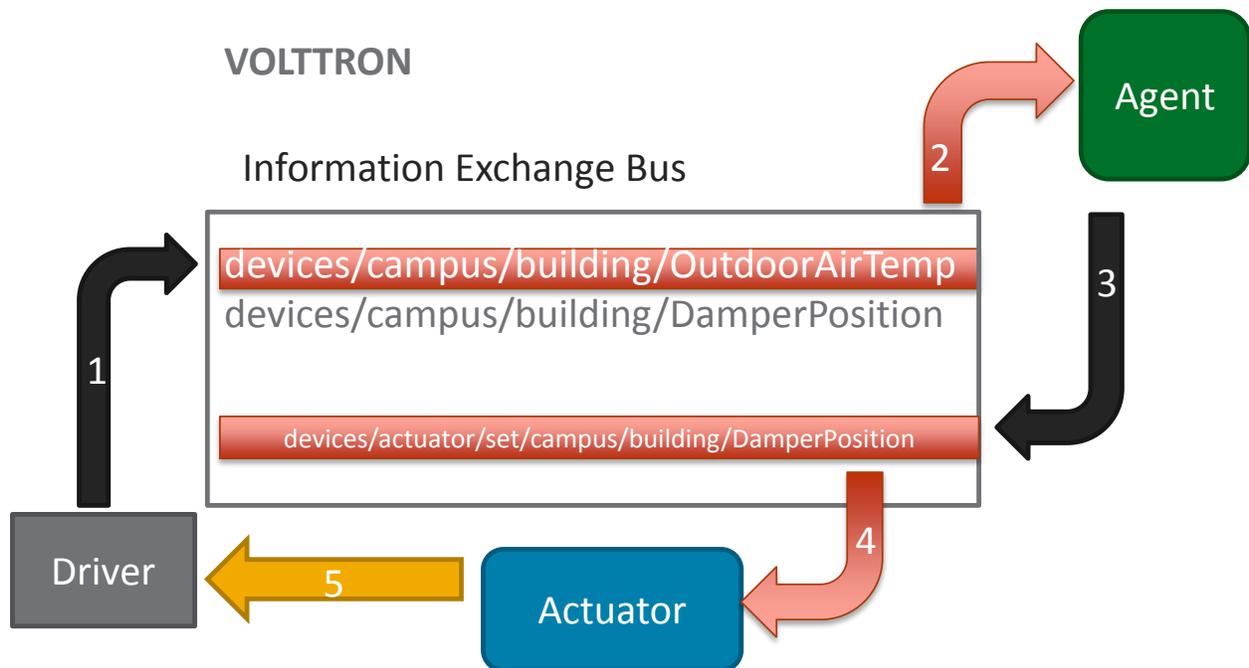


Figure 3.4. Diagram of an agent receiving data from and issuing commands to a device

An essential service of the platform is providing an integration point for various entities working together to achieve the goals of Transactional Energy. This integration service should hide the details of interfaces and protocols and provide a single point of contact. As long as an entity can communicate over this service, it can participate in the platform. The IEB component implements this service, as shown in Figure 3.5.

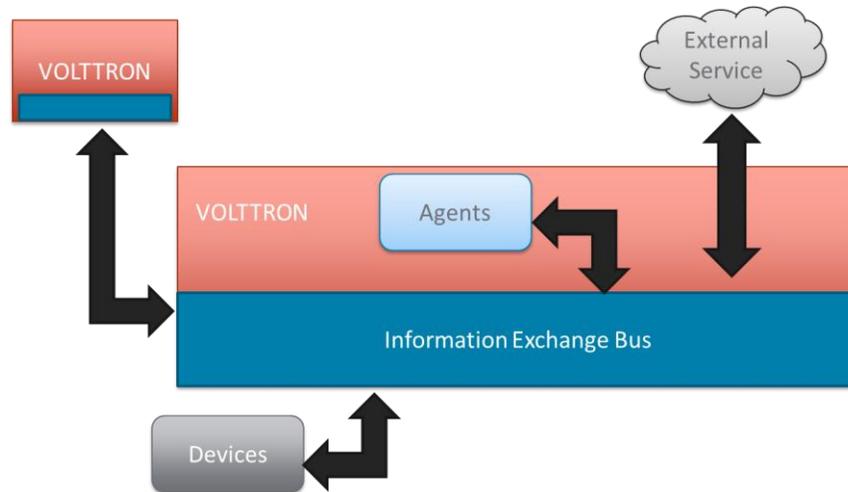


Figure 3.5. All actors involved in a VOLTTRON™ deployment communicate via the IEB

IEB Topics: The primary mode of communication between agents, agents and services, and any component on the TBC platform is through the IEB. The TBC platform provides a way for various components to communicate with each other using a publish/subscribe mechanism in the IEB. This allows for greater flexibility because topics can be created dynamically, and the messages sent can be any format as long as the sender and receiver understand it. For example, an agent with data to share publishes to a topic, then any agents interested in that data subscribe to that topic. While this flexibility is powerful, it could also lead to confusion if some standard is not followed. The following is an example structuring for the conventions for communicating in the TBC platform:

- Topics and subtopics follow the format: topic/subtopic/subtopic.
- Agents/services (subscribers) can subscribe to any and all levels. Subscriptions to "topic" will include messages for the base topic and all subtopics. Subscriptions to "topic/subtopic1" will only receive messages for that subtopic and any children subtopics.
- All agents should subscribe to the "platform" topic. This is the topic the TBC platform will use to send messages, such as "shutdown." This functionality can be handled in a base agent class that all others extend.

Messaging should also support headers for including metadata about the messages being sent on topics. For instance, agents should set the "From" header. This will allow agents to filter on the "To" message sent back. Platform service agents can utilize this information to direct replies to the requestor and allows agents to filter out messages not intended for them.

3.3.1.2 Command Module (annotated as component “2” in the Figure 3.1)

The Command Module shown in Figure 3.6 provides a command-line interface for controlling the platform functions such as maintenance and agent lifecycles. It allows the administrator to start and stop agents, set them to autostart with the platform, send them to another platform, etc. This service is built into the platform and utilizes user account security to ensure that only the account that started the platform can send commands to it.

```
hardware2@hardware2:~/workspace/volttron$ bin/volttron-ctrl list-agents
AGENT          AUTOSTART  STATUS
ev2.agent      enabled    running [2591]
multicomm.service enabled    running [2590]
```

Figure 3.6. Example of command-line output

Example commands available through this interface include:

- install: install agent from wheel
- tag: set, show, or remove agent tag
- remove: remove agent
- list: list installed agent
- status: show status of agents
- clear: clear status of defunct agents
- enable: enable agent to start automatically
- disable: prevent agent from start automatically
- start: start installed agent
- stop: stop agent
- run: start any agent by path
- shutdown: stop all agents
- send: send mobile agent to and start on a remote platform

3.3.1.3 Authentication and Authorization Module (annotated as component “3” in the Figure 3.1)

All agent software that executes on VOLTTRON™ must be validated before execution is allowed in order to meet the cyber security requirements of the platform. All data carried or produced by agents must be protected against tampering. In addition to these authentication tasks, activities performed by agents must be authorized according to the policies instituted by the electric utilities. For example, a metering agent on a smart meter may not have a need-to-know to access data related to power quality measurements or network authentication credentials.

A more detailed discussion of the security features of VOLTTRON™ can be found in section 3.6

3.3.1.4 Resource Monitor Module (annotated as component “4” in the Figure 3.1)

Agents perform both information sensing and control tasks. To execute their tasks reliably, the agents will require computational resources. If the platform cannot provide guarantees for available processing power, memory, or storage, then the agents may not be able to complete their tasks. On the other hand, if an agent consumes too much memory or processing power, it will adversely impact other agents or even the base platform functions. Therefore, a platform that provides resource guarantees for reliable execution of agents is highly desirable.

Agents will present the platform with an execution contract specifying their requirements. These requirements can be the specifics of the systems (operating system, required libraries, controlled devices, etc.) or system resources such as memory and CPU utilization. If the platform cannot support these requirements, the agent is rejected and not allowed to execute on the platform. If the agent was attempting

to move onto the platform via the Mobility Module, then a notice of rejection is sent back to the sending platform and the agent there.

3.3.1.5 Agent Mobility Module (annotated as component “5” in the Figure 3.1)

The Agent Mobility Module (implemented as a platform agent) allows agents to move between platforms. This service can be used by administrators to provision platforms in the field by sending them the latest version of agents needed for their deployment (see Figure 3.7). This service can also be utilized by agents to move between platforms as they accomplish some task. For instance, an agent could move from platform to platform collecting configuration information then return to the administrator to deliver the data (see Figure 3.8). Agents use this service by publishing requests to the message bus. If their request to move is denied by the receiving platform, they are notified via another topic on the bus and can attempt to move to another platform.

The different aspects such as packaging, transport, determination of destination, and validation of agents are handled by the Mobility Module in concert with the AIP (see Figure 3.7 and section 3.3.1.7). This keeps the agents as simple as possible and requires them only to publish a move request. Agents can carry data with them during moves in their immutable luggage. Agent packages are discussed further in section 3.4.2.6.

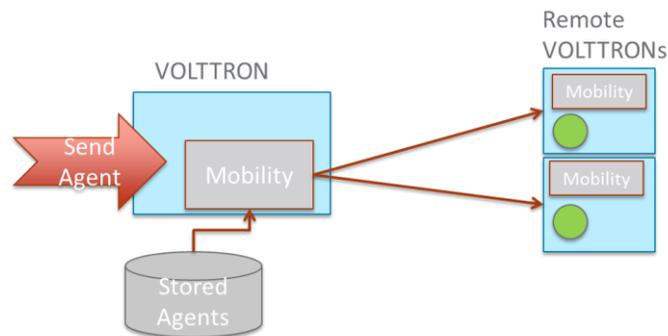


Figure 3.7. Administrator provisions remote VOLTRON™ instances by sending prepared agents (represented by green dots) to them

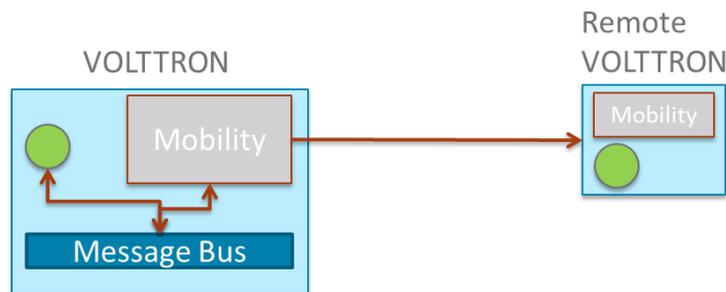


Figure 3.8. A running agent moves to a remote platform by sending a request to the Mobility Module

Detail on the operation of the Mobility Module can be found in section 3.4.2.4.

3.3.1.6 Multi-Platform Communication Module (annotated as component “6” in the Figure 3.1)

Communication between agents on separate VOLTRON™ platforms is enabled via the Multi-Building Communication Module (a platform agent). This service allows agents on one platform to publish and subscribe to the message bus of a remote platform as if they were local to it. This approach minimizes the communication between platforms as only data of interest is exchanged. The service listens on the local message bus for requests from agents to publish or subscribe to the message bus of a remote platform. It then forwards those requests to the MultiNode service platform (see Figure 3.9). This service then interacts with its local message bus to complete the request.

This service is optional and can be enabled/disabled by simply enabling/disabling the multi-platform service agent. It is easily configured using the service launch file and provides several new topics for use in the local agent exchange bus.

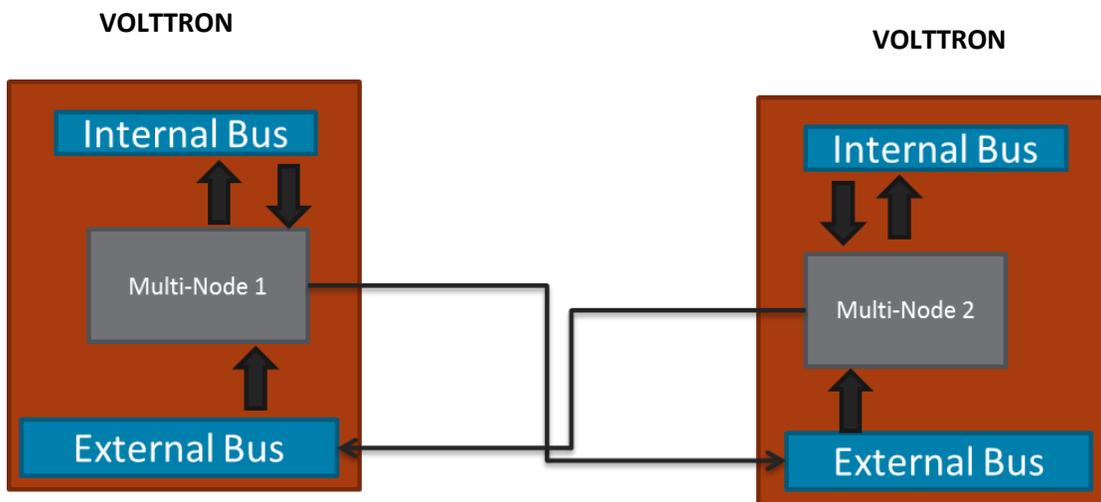


Figure 3.9. Communication between VOLTRON™ platforms via the MultiNode service

3.3.1.7 Agent Instantiation and Packaging (annotated as component “7” in the Figure 3.1)

The AIP module is responsible for receiving agent packages then utilizing the Authorization Module and Resource Monitor to verify the package and ensure the platform has the resources needed to execute the agent. The AIP also creates the AEE for the agent when launching. The Mobility Service interacts with the AIP to package agents for transport.

The discussion of the agent lifecycle in section 3.4.2 details the workflow of the AIP.

3.3.1.8 Agent Execution Environment Module (annotated as component “8” in the Figure 3.1)

The platform launches agents in an AEE that isolates them from the rest of the system. This prevents agents from writing to arbitrary locations on the file system and allows for resource monitoring and

limitation. The Resource Monitor works with the AEE to ensure that agents do not exceed the resource limit specified in their execution contract.

The AEE sets up the environment for the agent processes and sends necessary configuration information to communicate with the platform (socket for communicating with the message bus). If the agent extends the base agent, any platform specific parameters are dealt with automatically.

3.3.1.9 Actuation Module (annotated as component “9” in the Figure 3.1)

This platform agent acts as the intermediary for other agents sending control commands to devices in the platform. The Actuator works with the Scheduler to ensure that only agents with a valid device reservation can control devices (see Figure 3.10).

Agents who wish to send commands to devices first publish a request to the Actuator. The Actuator then checks for a valid reservation before forwarding the command on to the driver (section 3.3.1.10) interfacing with that device.

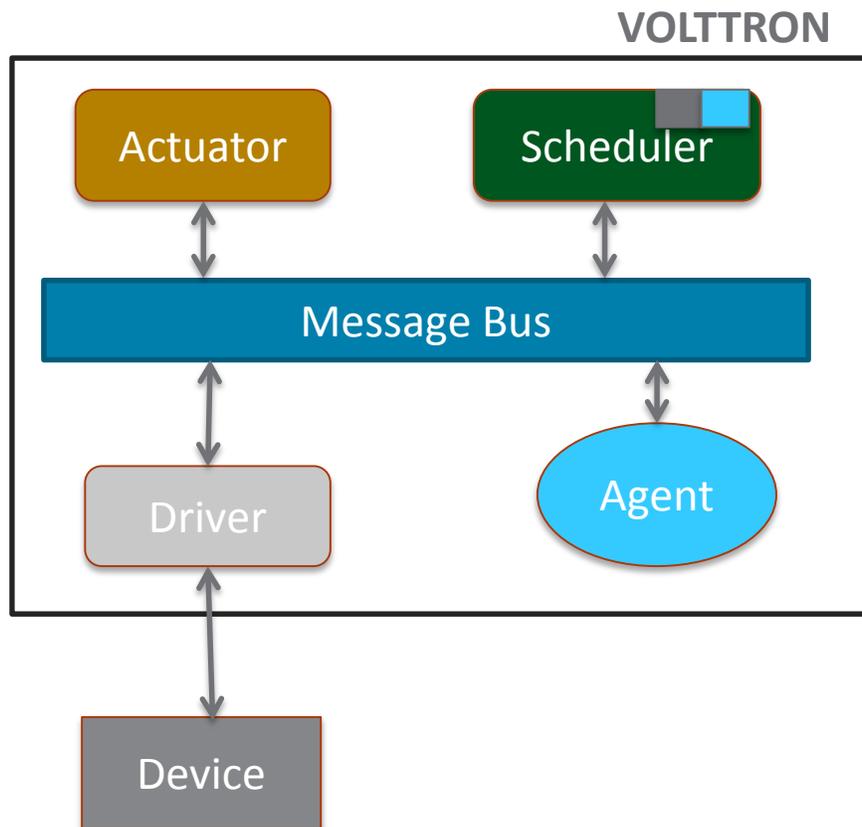


Figure 3.10. The Actuator coordinates agent control requests with driver agents and the scheduler

3.3.1.10 Drivers (annotated as component “10” in the Figure 3.1)

Drivers in VOLTRON™ are responsible for publishing data to the message bus and providing an interface for the Actuator for use in responding to command requests from agents (see Figure 3.11).

Drivers handle the details of talking the appropriate protocol to devices being controlled (BACnet, Modbus, custom, etc.) Drivers can be setup to work with a single device at a time or control multiple devices.

VOLTTRON™ drivers can be written as Platform Agents or specialized code that communicates with the IEB. The driver periodically publishes data on the message bus, where agents can then subscribe to some or all of the readings. Agent developers only need to specify the topics of interest to receive data instead of developing their own interfaces.

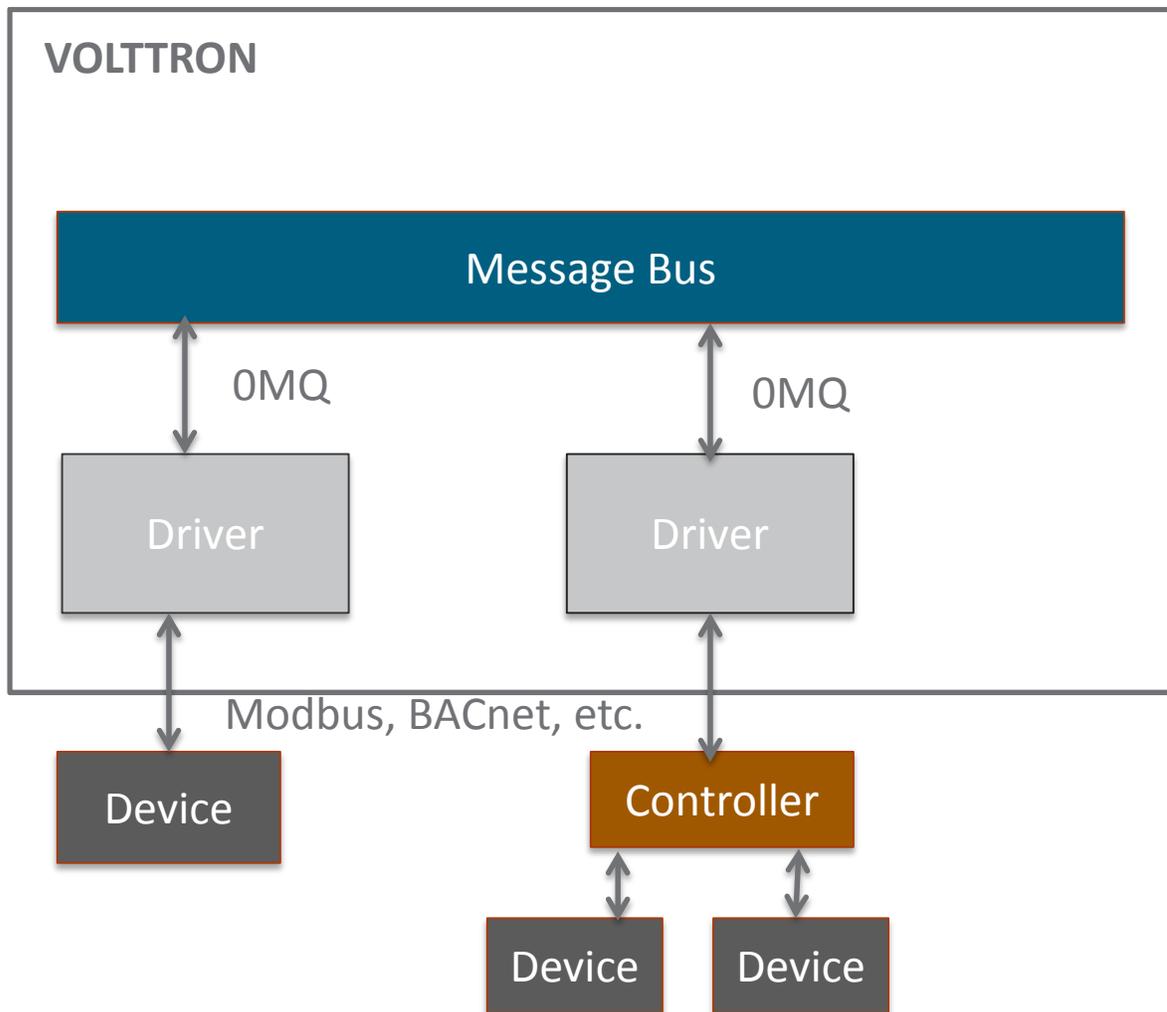


Figure 3.11. VOLTTRON™ drivers are the interface between devices and the platform. They can talk directly to devices or through a controller device.

Controlling Legacy Devices: To control legacy end-use devices, the VOLTTRON™ platform could include a controller that allows monitoring, control, and adjustments of devices, preferably in real time. The control device could be fully or partially embedded on the end-use device or be server-based and provide sensor information to the platform. A controller example is the Catalyst² controller used for

² <http://transformativewave.com/CATALYST>

HVAC systems. This controller can be retrofitted onto HVAC units, allowing the platform to interact with devices it would otherwise be unable to easily control. The Catalyst also contains its own algorithms and logic for operating the HVAC when no agents are performing actions. The MODBUS driver would be used to support communication and control of the Catalyst controller, as well as other control applications.³

3.3.1.11 Management Console Module (annotated as component “11” in the Figure 3.1)

The Management Console (Figure 3.12) gives administrators a view into the status of platforms under their control. This allows a non-expert user to be able to work with the platform without limiting them to a command-line type interface. Additionally, it must give “at-a-glance” status information for admins in charge of large numbers of VOLTTRON™ platforms.

Whereas the Command Module (section 3.3.1.2) is a command-line interface that requires logging onto the machine hosting VOLTTRON™, the Management Console allows access to all VOLTTRON™ instances in a deployment.

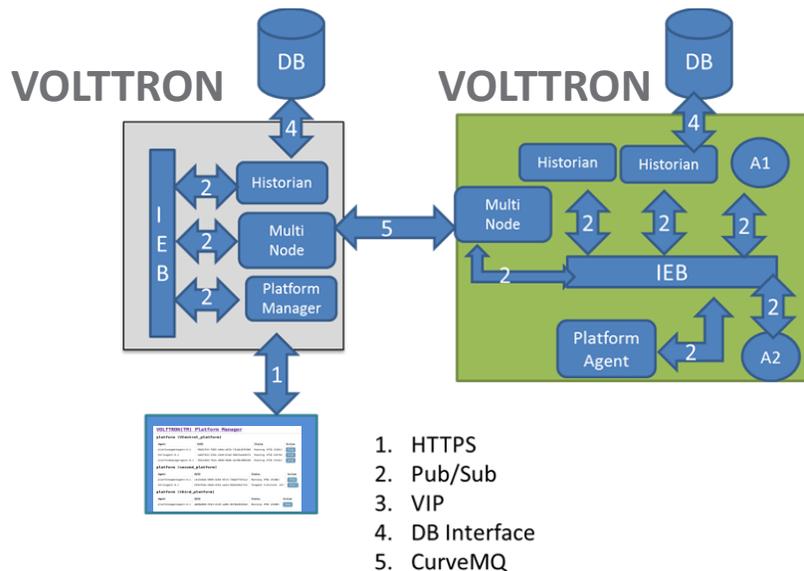


Figure 3.12. Management Console exposed as a web interface for use by administrators

3.3.1.12 Supervisory Agent (annotated as component “12” in the Figure 3.1)

The Supervisory Agent is a platform agent that provides an interface for reporting the status of the platform, its agents, and its devices. When combined with a Management Console, this provides

³ The MODBUS driver described in the Platform Infrastructure section is written to be as generic as possible and should handle most types of data including: shorts, long ints, floating point, double, and string. In the context of the sMAP driver, it is restricted to all variants of integers and floats because the historian cannot handle any other types. Certain devices may use a byte-ordering scheme, which is incompatible with this service. For instance, the Dent power meter uses missed Endian types (byte ordering), which cannot be handled in a generic way. Additional code will be required for these special cases

administrators with insight into the state of the platforms under their control. This agent also responds to commands from the Management Console.

3.3.1.13 Historian (annotated as component “13” in the Figure 3.1)

Historian agents subscribe to all data being published from devices or logged from agents and pushes them to permanent storage for later retrieval (see Figure 3.13). Historian agents abstract the details of this storage and can be implemented for a variety of solutions: local file/database, remote service, etc.

Historians listen on an agent logging topic and push data from this to the external historian using topics specified by the agents. These topics can be dynamically created as needed. For example, if the source name is 'test data', and the topic published is `datalogger/log/campus1/building1/testdata`, then the data will be posted to the time series under `campus1/building1/testdata` in the source name 'test data'.

Each specific historian (SQLite, MySQL, file based, etc.) extends a base historian that simplifies their development and maintains a local cache of data to ensure no data are lost in case communication with the external server fails. The specific implementations do not need to handle this case; they simply write the cached data when able.

The platform also provides query service that allows agents to specify a data point and time range to receive historical data. Agents can use this mechanism to build a baseline of building sensor readings before making control decisions.

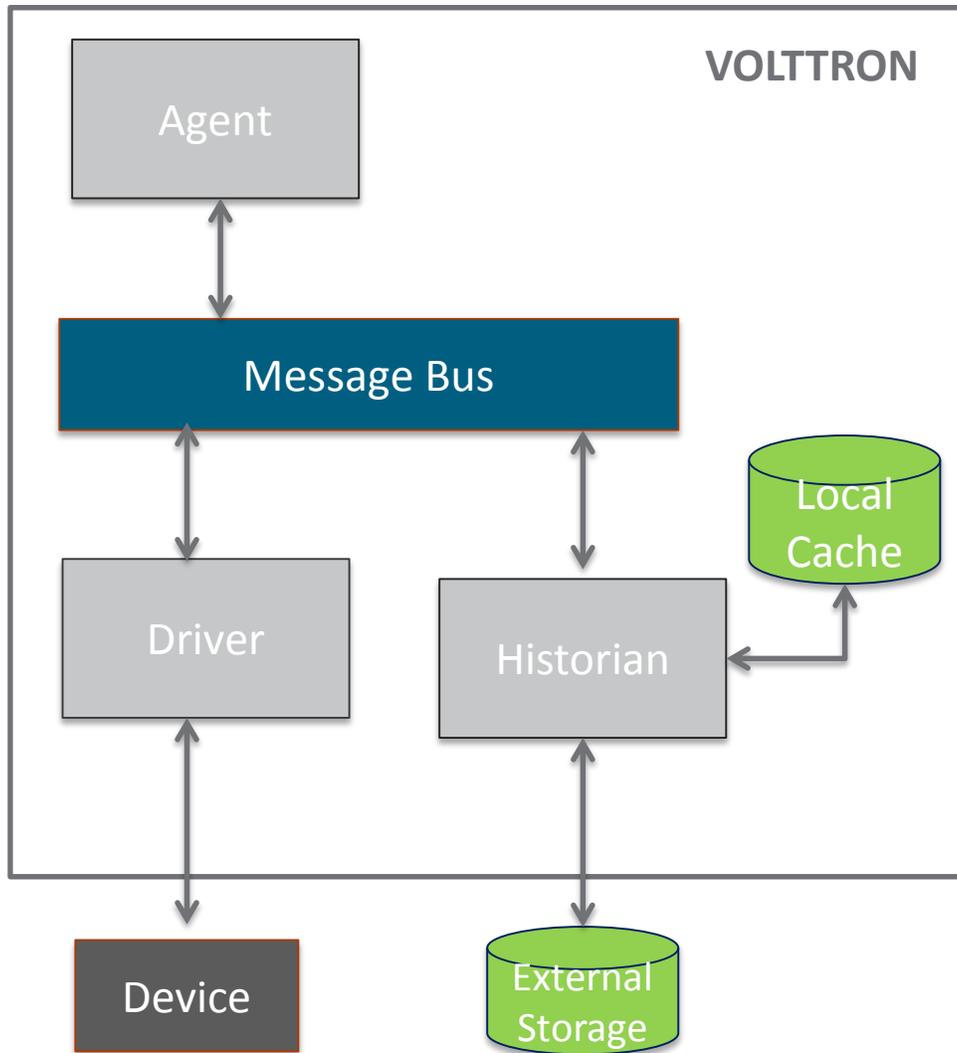


Figure 3.13. Historian agents store data from the message bus in a local cache until successfully writing to external storage

3.3.1.14 Scheduler Service

The platform provides a scheduling service allowing agents to reserve devices during a time slot. This ensures agents that need access to a device for an experiment will not be locked out by other agents during the duration of the test. Reservations can be made in advance and agents can indicate a priority. Low priority reservations may be preempted by those with higher priority. In cases where requests are denied, the service should report which agent(s) has the desired device that would enable agents to negotiate amongst themselves for control of devices. The scheduling process is outlined in Figure 3.14.

Agents must propose a schedule that is accepted if there are no conflicts with other agent requests. In cases of conflict, the agent is informed which agent already has a reservation. This opens up the potential for agents to negotiate with each other for control over devices.

Additionally, this service can be utilized as an external task scheduler by allowing agents to make reservations on non-device topics. This allows agents to go into a sleep mode until their timeslot is announced allowing them to wake up and resume execution.

Tasks should indicate priorities to enable important and/or time sensitive actions to preempt low importance/non-sensitive reservations. Suggested task priorities include:

- **HIGH:** This task cannot be preempted under any circumstance. This task may preempt other conflicting pre-emptible tasks. For one device there can be only one HIGH priority task.
- **LOW:** This task cannot be preempted once it has started. A task is considered started once the earliest time slot on any device has been reached. This task may not preempt other tasks.
- **LOW_PREEMPT:** This task may be preempted at any time. If the task is preempted once it has begun running, any current time slots will be given a grace period (configurable in the Actuator Agent configuration file, defaulted to 60 seconds) before being revoked. This task may not preempt other tasks.

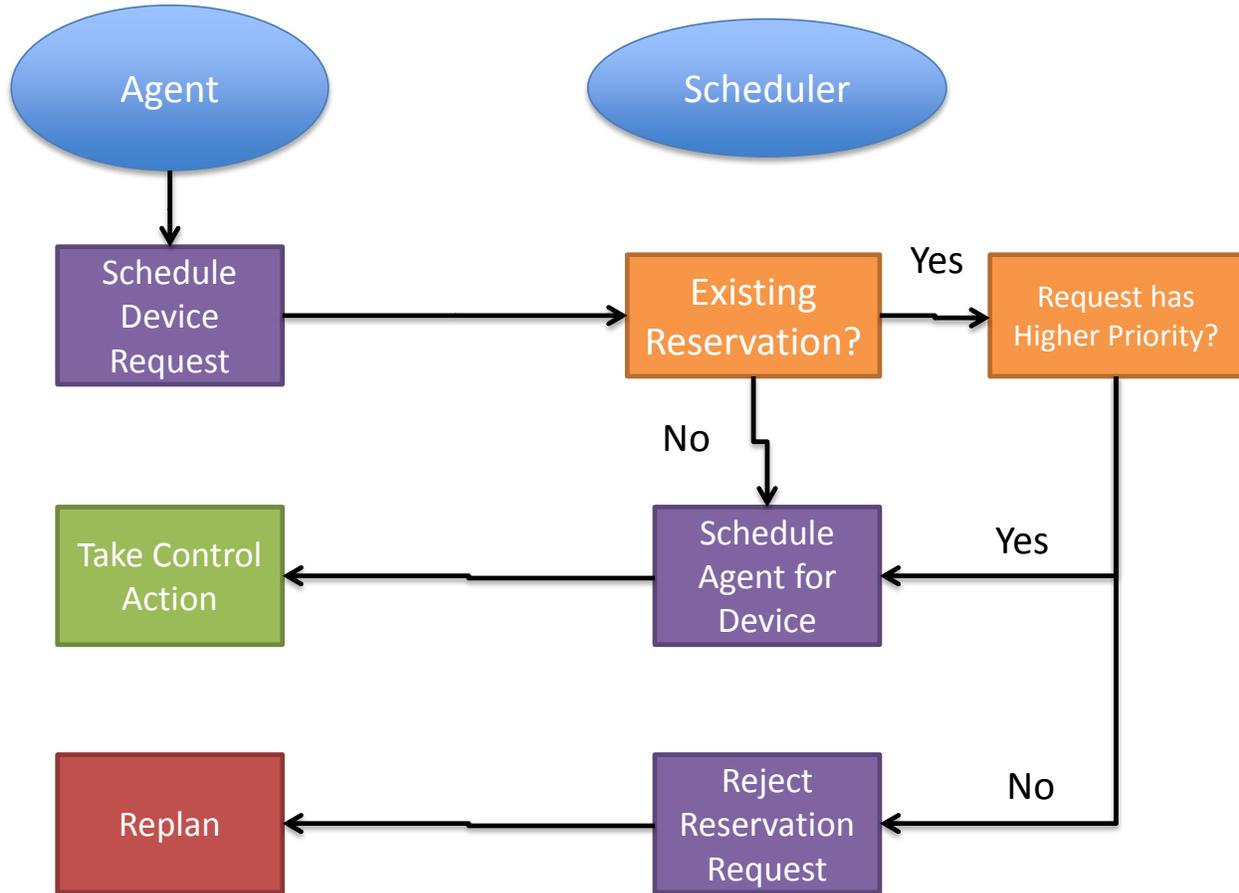


Figure 3.14. Agent requesting control for a device

This service must maintain its state in cases where it or the platform itself goes down. Upon restart, any existing reservations must still exist. The service should also periodically announce the current schedule so agents can plan their actions accordingly.

3.3.1.15 Weather Service

As a Cloud agent, the Weather Service agent shown in Figure 3.15 publishes weather data from a site such as Weather Underground to the platform. It retrieves data based on a zip code setting in its configuration file and periodically publishes data to a weather topic on the platform. It can also listen for data requests from agents for instant requests for data instead of waiting for the next publish time.

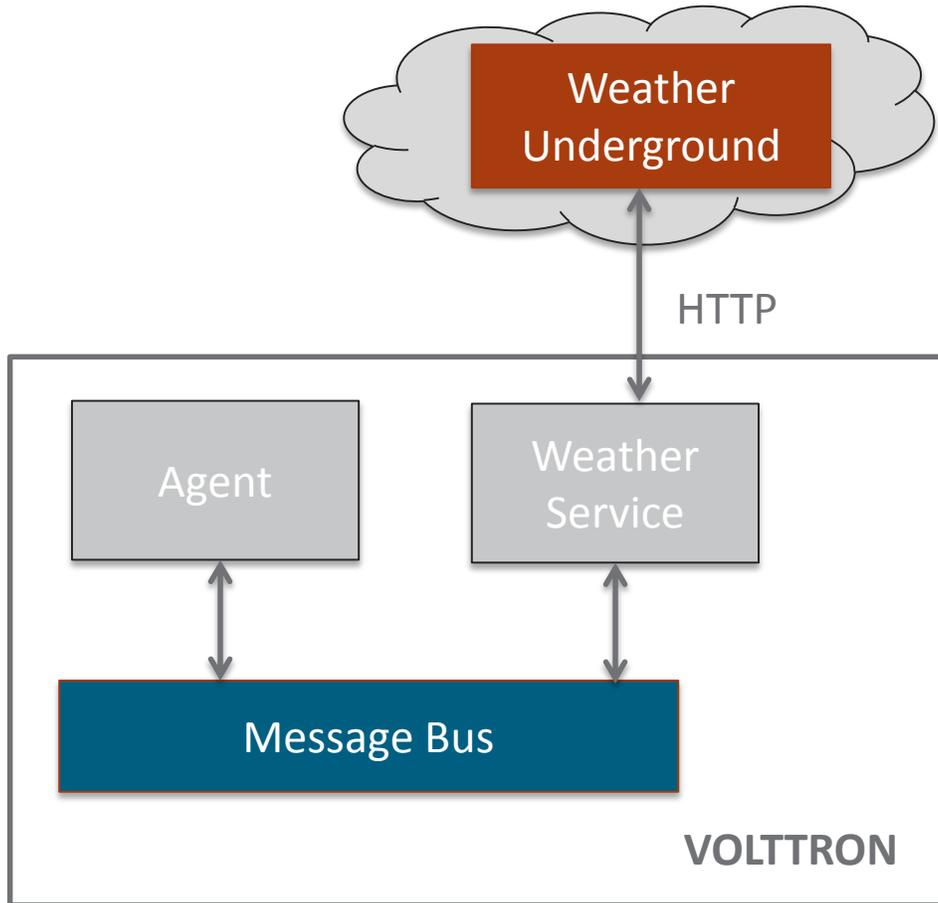


Figure 3.15. The Weather Service acts as a proxy allowing agents to interact with a web service via the message bus

3.3.1.16 Directory Service

Directory services are used for accessing credentials in an efficient and automated manner and for determining nearby resources and devices. Use of directory services allows users to avoid hard-coding information into deployed devices and supports the mobility and assignment of tasks to agents. The directory services built into the TBC platform should be robust and use peer-to-peer communication.

3.4 VOLTTRON™ Platform Details

VOLTTRON™ provides a set of services that enables application developers to work with devices, utilities, other applications, etc. without being tied to the details of those systems. VOLTTRON™ also

ensures that devices are not given multiple conflicting commands and can be returned to a safe state in case of a fault.

The following section describes in general terms the services and components of VOLTTRON™ as an example platform built to meet these requirements supplementing the material provided above.

3.4.1 VOLTTRON™ Platform Services

VOLTTRON™, as a TBC platform, provides various services that support higher-level objectives and that can be shared by more than one functional application. These tend to be general utility services, such as to access databases containing local information or information obtained through remote third-party services. But they may also be higher-level services that implement behavior models, optimize and forecast algorithms, measure and verify services, or perform isolated functions such as settling payments. What distinguishes these entities as services in this architecture is that they may be invoked by multiple higher-level applications, with their shared access being managed by the platform's core infrastructure.

A key function of the platform is supporting the lifecycle of the agent. A discussion of how the VOLTTRON™ components described in section 3.3 interact to support the agent lifecycle follows.

3.4.2 Agent Lifecycle

Agents running in VOLTTRON™ utilize different services for each portion of their lifecycle. In a typical lifecycle, agents are created by one organization then deployed on platforms being operated by another. Another entity could then monitor and upgrade these agents as needed. This section discusses various processes associated with the life of an agent from creation to destruction.

3.4.2.1 Agent Creation

Agent Creation is the process by which VOLTTRON™ vendor or developer creates a new agent. First, the vendor or developer develops an agent for the VOLTTRON™ platform. The code for the agent is then cryptographically signed by its creator (e.g., vendor) at creation time. As part of the agent creation, the creator also defines the resources (such as processing and memory required for the agent to execute) and attaches the execution requirements to the agent code. Execution requirements are referred to as the *agent execution contract*. The creator also signs the completed agent package containing the code and the agent execution contract. The complete agent package (referred to in the text as the agent transport payload [ATP]) needs to be validated by the administrator of the organization deploying the agents to ensure that the agent code is not tampered with by a third-party entity.

3.4.2.2 Agent Initiation

The Agent Initiator initiates a new agent and sends it out to devices via the Mobility Service. After the Initiator has prepared the ATP for a specific deployment, the Initiator's signs the ATP to affirm that it has validated the organizational administrator's signature on the stored agent and the administrator's role as an authorized organizational administrator (internal or external). The Initiator's signature enables the recipient to validate that the Initiator-signed items have not been modified since the Initiator signed them.

3.4.2.3 Agent Execution

Agent execution includes a set of tasks including verification of the authorization and authenticity of the agent prior to actual execution. Once execution is complete, data are collected, a new path is selected, payload is reconstructed, and the agent moves to another device. Figure 3.16 outlines the agent execution workflow.

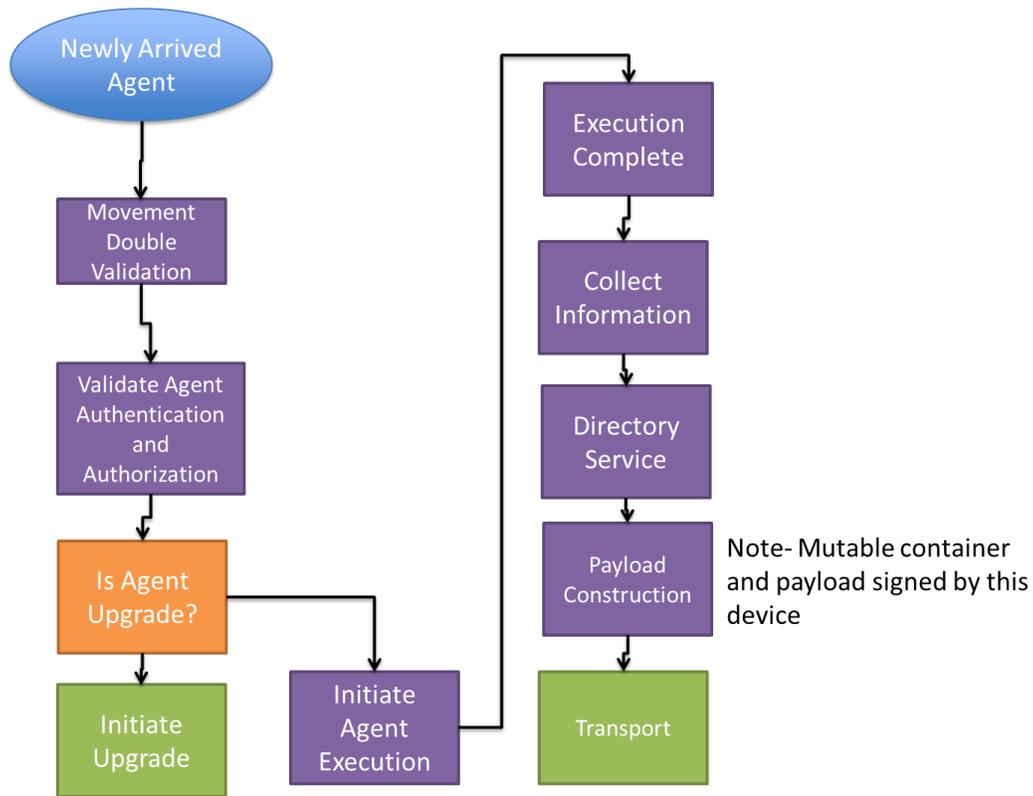


Figure 3.16. Workflow for Agent Execution

3.4.2.4 Agent Movement

Agents can request to move between platforms after a task is complete on their current platform. This allows agents to visit multiple VOLTTRON™ hosts and then move on and free up resources. Especially in small embedded devices, this prevents unneeded agents from taking up space and resources. It also allows for a simple upgrade path by sending out new versions of an agent to perform the next set of tasks. Workflow for agents moving between platforms can be seen in Figure 3.17.

In a typical scenario, the agent either makes a move request or is commanded to move by the administrator. The agent is first packaged for movement and the source signature is added as discussed in section 3.4.2.6. The execution contract can then be sent in advance to the target platform where it can be evaluated. If the target platform can support the agent, then the agent package is sent. Once received by the target platform, the agent is then started. If both the attempt to send and the attempt to start the agent succeed, the sending platform terminates the agent and could remove it to save storage.

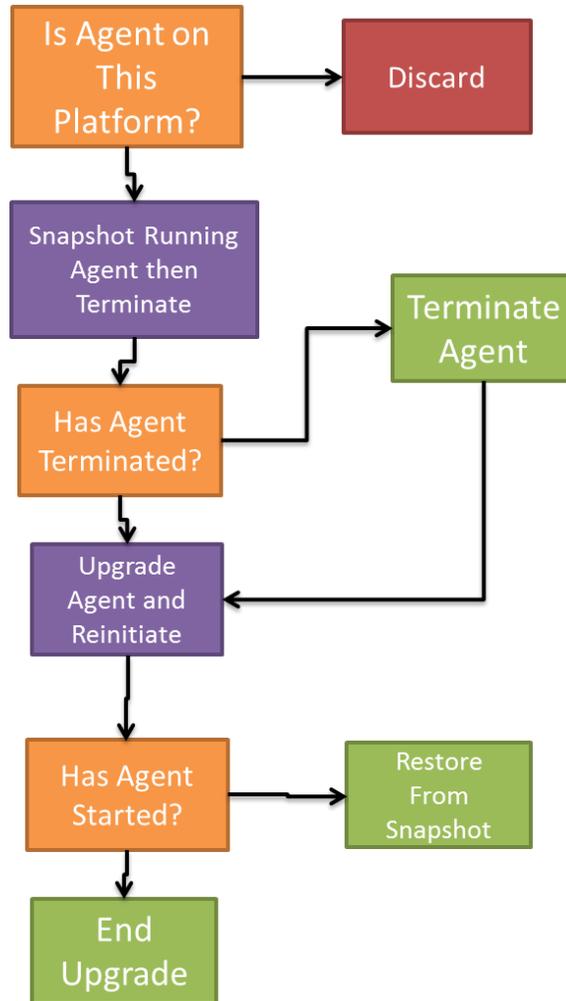


Figure 3.18. Workflow for Agent Upgrade

3.4.2.6 Agent Transport Payload

The ATP is used to ensure that agents are not tampered with while moving between platforms or between launches on the same platform. Each layer described in the ATP is signed by an entity that signifies they have examined the code for security and correctness. Figure 3.19 outlines the components of the ATP. A description of the components is provided below.

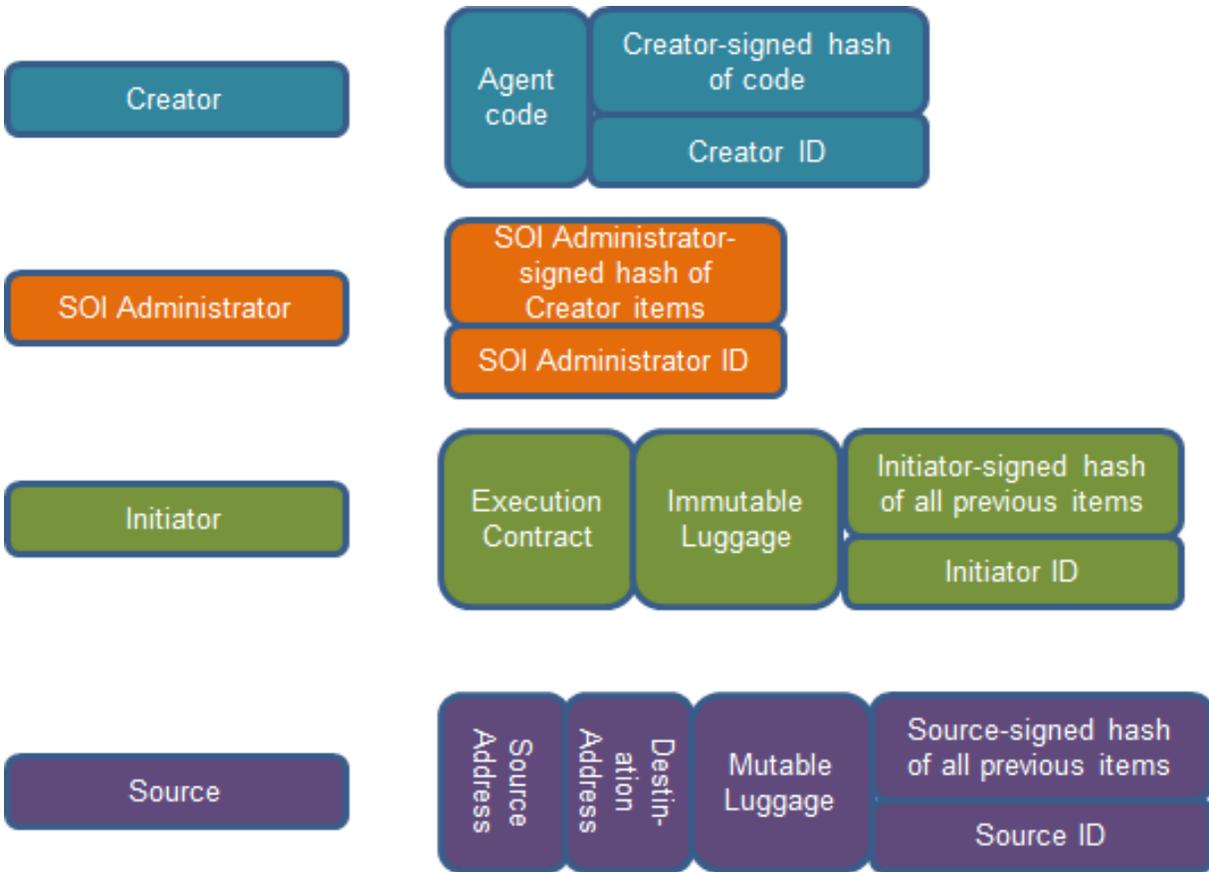


Figure 3.19. Components of the Agent Transport Payload

- **Creator ID:** The entity that created the agent code. The Creator does not actually instantiate the agent. Creator ID is a “name” in the agent namespace that can be further mapped by using directory services to a set of unique identifiers such as a network address.
- **Agent Code:** Actual “executable” code for the agent that will be used by the AEE. Based on the AEE, the code can vary from an executable script to Java byte code.
- **Initiator ID:** The device that instantiated the agent. This device is responsible for setting configuration parameters, tasking the agent, etc. The Initiator does not modify the code in any way but adds the configuration and tasking information to the configuration data that is part of the immutable luggage. Initiator ID is a “name” in the agent namespace that can be further mapped by using directory services to a set of unique identifiers such as a network address.
- **Execution Contract:** The contract the agent makes with the platform onto which it is moving. This contains the resources (memory and CPU) required to execute and the actions it wishes to perform. The resource component is used by the Resource Monitor while the actions section is used by the Configuration Policy module. This should also contain the version of the platform this agent was written against. A compatibility check needs to be run as part of the validation process.
- **Immutable Luggage (Configuration Data):** Any task configuration parameters, such as the agent path (the set of nodes to be visited, not necessarily sequentially), initial data, payloads, etc. needed for the

instantiation and proper running of the agent. This information cannot be changed during the lifecycle of the agent.

- **Destination ID:** The device onto which the agent is moving. Destination ID is a “name” in the agent namespace that can be further mapped by using directory services to a set of unique identifiers such as a network layer address.
- **Source ID:** The device from which the agent moved. Source ID is a “name” in the agent namespace that can be further mapped by using directory services to a set of unique identifiers such as a network address.
- **Mutable Luggage:** Any current state, data, payload, etc. that the agent needs to perform its task. This can be changed at every stop the agent makes. Optionally, mutable luggage may contain personally identifiable information or business sensitive data that may need to be encrypted. Therefore, mutable luggage needs an indicator to identify whether the data is encrypted. It is recommended that each host provide a signature to attest that the agent has been there and was run there. The signatures can be checked by the initiator when the agent returns after completing its mission to determine if any devices in the path were skipped.
- **Validation Sub-Payload(s):** Used to validate that each sub-payload of the ATP is authentic and was signed by an entity with the correct role. The validation sub-payload(s) include a cryptographically signed hash and the ID of the signer to enable the recipient to validate that the signed portion of the ATP has not been tampered and to validate that the signer was authorized to sign the payload. The entity that signed as the source device should also be compared to the actual source device to ensure that they are the same.

3.5 Applications Built Using the VOLTTRON™ Platform

Applications are implemented by one or more agents to meet specific functional objectives (Figure 3.20). VOLTTRON™ potentially hosts multiple such applications running—likely concurrently—within the platform. A typical example of an application running on VOLTTRON™ within a building would be one for managing and controlling the energy consumption of end-use devices on the site—that is a collection of agents, implementing an application, and performing some or all of the typical functions of a building automation system (BAS) (as shown by green “V” in the Figure 3.20). In addition, another platform may run applications to provide specific transactional service participation (a VOLTTRON™ instance running on the business network as shown in Figure 3.20). Ideally, in the implementation for a TBC platform, these particular applications would be decoupled so that the BAS applications function independently of the transactional service application, with the latter performing transactions that result in new constraints and parameters affecting the BAS activities.

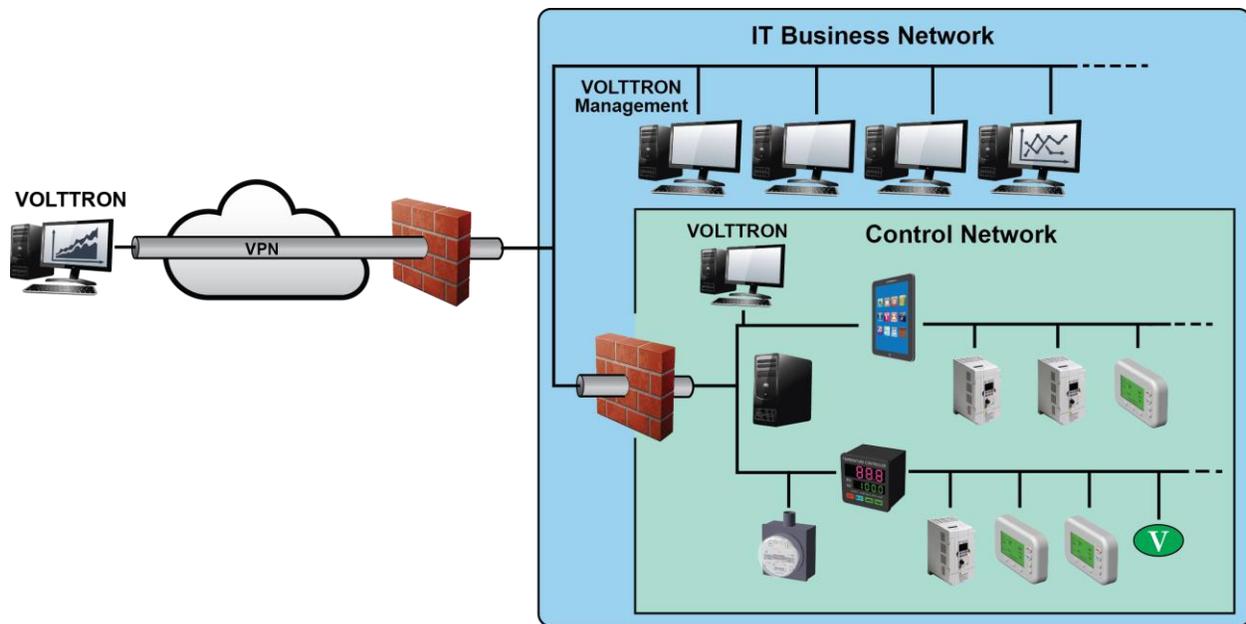


Figure 3.20. An Example Deployment of VOLTRON™ in an Enterprise

The following sections provide example applications and discuss how they interact with the TBC platform.

3.5.1 Example Energy Efficiency Agent/Application Deployment on TBC

As stated previously, a TBC platform can be used to deploy a number of transactional services, including energy efficiency and grid services. This section describes an example of how a TBC platform would deploy an energy efficiency and a grid service agent/application on the TBC platform and the TBC platform services the application would use.

3.5.1.1 Diagnostic Agent/Application

The automated fault detection and diagnostic (AFDD) processes can be used to identify operation faults/problems with heating, ventilation, and air conditioning (HVAC) systems. A number of possible AFDD process can be deployed on the TBC platform. The airside economizer diagnostic process for either an air-handling unit (AHU) or a rooftop unit (RTU) is used to illustrate how to deploy this type of a service on a TBC platform.

In an AHU or RTU, the airside economizer modulates controllable dampers to use cool outdoor air either to supplement or entirely displace mechanical cooling, when outdoor-air conditions are favorable. Unfortunately, economizers often fail and do not work properly, leading to increased energy use rather than saving energy. Common problems include incorrect control strategies, diverse types of damper linkage and actuator failures, and out-of-calibration sensors. These problems can be detected using sensor

data that are normally present on the AHU or RTU to control the system. For more details on the diagnostics method refer to Katipamula et al. (2014) and Lutes et al. al. (2014)⁴.

The AFDD agent utilizes many of the TBC platform services. The following section 3.5.2 provides details about how the AFDD and grid service agents utilize the platform services (which are shown in italics).

3.5.2 AFDD Utilization of Platform Services

The lifecycle of an agent can be divided into three primary components: 1) configuration prior to agent execution, 2) data collection, and 3) execution of the agent to create meaningful results (analysis, commands to devices, etc.).

Configuration: The TBC platform supports text-style configuration files that contain key (name of configurable parameter) value pairs of any datatype (often JSON-style configuration files are used). The TBC platform could also support a graphical front for agent configuration. Agent configuration allows for development of a generic application and then provides a means by which an end user can customize the application to meet their needs.

Data Collection: Like all agents, the primary means of exchanging information between agents and the various TBC platform services and devices is through the *IEB*. The AFDD agent subscribes to the relevant data that it needs and also publishes data relevant to other services and agents. The AFDD agent also utilizes the *IEB* to interact with and control HVAC devices.

The data needed for the AFDD agents are published by the *device interface service*, which gathers the data from the devices (e.g., AHU or RTU) and publishes with a specific device and specific topic on to the *IEB*. The AFDD subscribes to this topic(s) giving it access to the device data for use in its analysis. Passive diagnostic agents that do not require control of a device to carry out their function rely solely information from the *IEB* and *device interface service*. The AFDD agents can be run either in the passive or proactive mode.

Execution: Execution of the AFDD agent makes use of several platform services during its execution:

- **Actuation (or control):** In the proactive mode, the AFDD process involves commanding the HVAC devices to transition into different modes of operations. A proactive diagnostic approach allows for simulating operating conditions that may not occur for some time, thus producing results that otherwise might not be available for months. The AFDD agent accomplishes this by actively controlling a device (AHU or RTU). Actuation (or control) points for the device are described in the configuration file for the *device interface service* (Modbus, BACnet, etc.,) and actuation of these points is accomplished by publishing commands to the “actuator” topic on the *IEB*.

⁴Katipamula S, RG Lutes, H Ngo, and RM Underhill. 2013. Transactional Network Platform: Applications. PNNL-22941, Pacific Northwest National Laboratory, Richland, WA.

Lutes R.G., J.N. Haack, S. Katipamula, K.E. Monson, B.A. Akyol, B.J. Carpenter, and N.D. Tenney. 2014. VOLTTRON™ 2.0: User Guide. PNNL-23879, Pacific Northwest National Laboratory, Richland, WA

- **Scheduling Controls:** Access for active control of HVAC devices is managed by the platform *scheduling service*. This service prevents multiple agents from sending commands to the same device at the same time, which could cause undesirable or even erratic device behavior. The AFDD utilizes the platform *scheduling service* to reserve a time slot for device interactions. The AFDD publishes a device interaction request on the platform *IEB*. This request will identify which device the agent wishes to interact with, the desired time for the request, and the request priority (low and pre-emptible, low, or high). The AFDD utilizes a *low and pre-emptible priority*; if preempted during the diagnostic process, the AFDD will return the HVAC system to normal operating conditions and release control of the device to the preempting agent. The platform *scheduling service* will receive the device interaction request, determine the availability of the device (i.e., has another agent already requested this time slot), and will respond to the device interaction request on the actuator “result” topic with either success, the device interaction is scheduled, or failure. When a failure message is returned by the *scheduling service*, it is accompanied by information as to why the request was not accepted. This information can be used by the agent to decide how to deal with failure of a device interaction request.
- **Agent Execution:** As the AFDD executes the diagnostics, it produces actionable results that describe the operational state of the HVAC system. These results are stored in the platform historian that uses a product called the Simple Measurement and Acuation Protocol (*sMAP*)⁵. The AFDD accomplishes this by publishing the diagnostic results to the “logging” topic on the *IEB*. The logging service will retrieve data published under that topic and post them to the platform historian. Once the AFDD results are posted to the historian, they can be accessed through the historian by an end user or by other agents by way of the platform *archiver service*.
- **Other:** The AFDD agent also utilizes the platform *weather service*. Validation of sensor data is one obstacle for fault detection of HVAC systems. HVAC systems in buildings typically lack analytical redundancies that make validation of sensor data especially difficult. The AFDD uses the platform *weather service* to create such redundancy and to validate the outdoor-air temperature sensor. Accessing the platform *weather service* is accomplished by publishing the desired zip code on the “weather/request” topic on the *IEB*. The weather service will respond by publishing detailed weather data for the desired zip code on the “weather/response” topic on the *IEB*.

3.5.3 Grid Service Agent/Application

Many utilities around the country have or are considering implementing dynamic electrical pricing programs that use time-of-use (TOU) electrical rates. TOU electrical rates vary based on the demand for electricity. Critical peak pricing (CPP) is an example of a TOU rate with two types of days: normal TOU days and critical peak days or event days. On normal days the rates changes by time of use. On CPP days, the utilities charge an increased price for peak hours. CPP times generally coincide with peak demand on the utility; these events are generally called between 5 to 15 times per year and occur when the electrical demand is high and the supply is low. Customers on a flat standard rate who enroll in a peak-time rebate program receive rebates for using less electricity when a utility calls for a peak-time event. This

⁵ <http://www.cs.berkeley.edu/~stevedh/smap2/>

implementation of the grid service agent is specific to the CPP, but it can easily be modified to work with other incentive signals (real-time pricing, day head, etc.).

The main goal of the building owner/operator is to minimize the electricity consumption during peak periods on a CPP day. To accomplish that goal, the grid service agent performs three distinct functions:

1. **Pre-Cooling:** Prior to the CPP event period, the cooling and heating set points are reset lower to allow for pre-cooling (to ensure the RTU is not driven into a heating mode). This step allows the RTU to cool the building below its normal cooling set point (3-5°F below the normal) while the electrical rates are still low (compared to mid-peak and peak hours).
2. **Event:** During the CPP event, the cooling set point is raised 4-5°F above the normal, the outdoor-air damper is commanded to a position that is slightly below the normal minimum (half the of the normal minimum), the fan speed is slightly reduced (by 10% to 20% of the normal speed, if the unit has a variable frequency drive), and the second stage cooling differential (time delay between stage one and stage two cooling) is increased (by few degrees, if the unit has multiple stages). These steps will reduce the electrical consumption during the CPP event. The pre-cooling actions taken in step one will allow the temperature to slowly float up to the CPP cooling temperature set point and reduce occupant discomfort.
3. **Post-Event:** The grid service agent will begin to return the RTU to normal operations by changing the cooling and heating set points to their normal values. Rather than changing the set point in one step, the set point is changed gradually over a period of time to avoid the “rebound” effect (A spike in energy consumption after the CPP event when RTU operations are returning to normal). Another possible option to reduce the rebound effect is to duty cycle the RTU in an intelligent way, which would ensure that not all RTU are on at the same time.

3.5.3.1 Grid Service Utilization of Platform Services

The grid service agent utilizes the configuration, data collection, and execution services described below.

Configuration: The grid service agent requires user configuration prior to execution. The grid service agent allows the user to specify the set points and HVAC component settings (minimum outdoor-air damper command, fan speed, etc.) during pre-cooling and the CPP event. This allows the user to specify the minimum level of temperature discomfort acceptable for occupants and to modify the controls of the devices within that range. Other factors such as zone priority or required load reduction can also be used. Configuration of the grid service agent is accomplished through a JSON-style text file but other input methods (graphical frontend) can be added if desired.

Data Collection: Like all agents, the primary means of exchanging information between agents and the various TBC platform services devices is through the IEB. The grid service agent subscribes to the relevant data that it needs and also publishes data relevant to other services and agents. The grid service agent also subscribes to utility signals provided by an OpenADR client agent. Utility signals (demand response) typically originate at a retail electric utility or a wholesale independent operator indicating a need for consumers to modify their electricity usage or shift usage to another time period. The OpenADR agent receives data from an OpenADR server. The OpenADR server is hosted by a third party (utility or

the independent system operator). The information received by the OpenADR client includes TOU rates, upcoming CPP event, and other utility information pertinent to the grid. The OpenADR agent publishes this information on the IEB for use by the grid service agent (and other agents or platform services). This signal contains information on the date and time for a CPP event. The OpenADR signal and the user configuration parameters allow the grid service agent to initiate its demand response actions.

Actuation (or control): The grid service agent actively modifies the zone set points and other HVAC-related controls to reduce load during a CPP event. Actuation (or control) points for the device are described in the configuration file for the device interface service (Modbus, BACnet, etc.,) and actuation of these points is accomplished by publishing commands to the “actuator” topic on the IEB.

Scheduling Controls: The grid service agent utilizes the platform scheduling service to reserve a time slot for device interactions. The platform scheduling service will receive the device interaction request, determine the availability of the device (i.e., has another agent already requested this time slot), and will respond to the device interaction request on the actuator “result” topic with either success, the device interaction is scheduled, or failure. When a failure message is returned by the scheduling service, it is accompanied by information as to why the request was not accepted. This information can be used by the agent to decide how to deal with failure of a device interaction request. The grid service agent utilizes a high priority status, indicating the grid service agent will take precedence over lower priority agents. Once the device interaction is scheduled, the grid service agent cannot be preempted by another agent.

Agent Execution: As the grid service agent executes during a grid or CPP event, it publishes a status to the logging topic to indicate the start and end of each of the CPP functions (pre-cooling, event, and post-event). These results are stored in the platform historian sMAP. The grid service agent accomplishes this by publishing the diagnostic results to the “logging” topic on the IEB. The logging service will retrieve data published under that topic and post them to the platform historian. Once these results are posted to the historian, they can be accessed through the historian by an end user or by other agents by way of the platform archiver service. This information could be used to assess the performance of the grid service agent in reducing electrical load during the CPP event by way of measurement and verification agent. This information could also be used to plan future reaction to grid events and modify the grid service agent’s reaction to such events (i.e., modify the grid service agent’s configuration parameters) to increase load reduction or increase human occupant comfort.

3.6 Security Features of VOLTTRON™

Figure 3.21 shows two VOLTTRON™ systems communicating with each other and with external sources. The system on the left side (Figure 3.21) is expanded to show internal components of VOLTTRON™ and communication with an external historian and a Cloud-based service. Each interface in the figure is numbered so it can be referenced in the text. The approach taken is to discuss security threats associated with each interface. VOLTTRON™ is built on top of a modern Linux operating system. While VOLTTRON™ addresses threats associated with the platform and its interfaces, equal thought needs to be given to the security of the underlying operating system. The VOLTTRON™ team relies on comprehensive security hardening of the operating system as well as keeping up to date with periodically applied patches to further enhance the security of the instances deployed in the field.

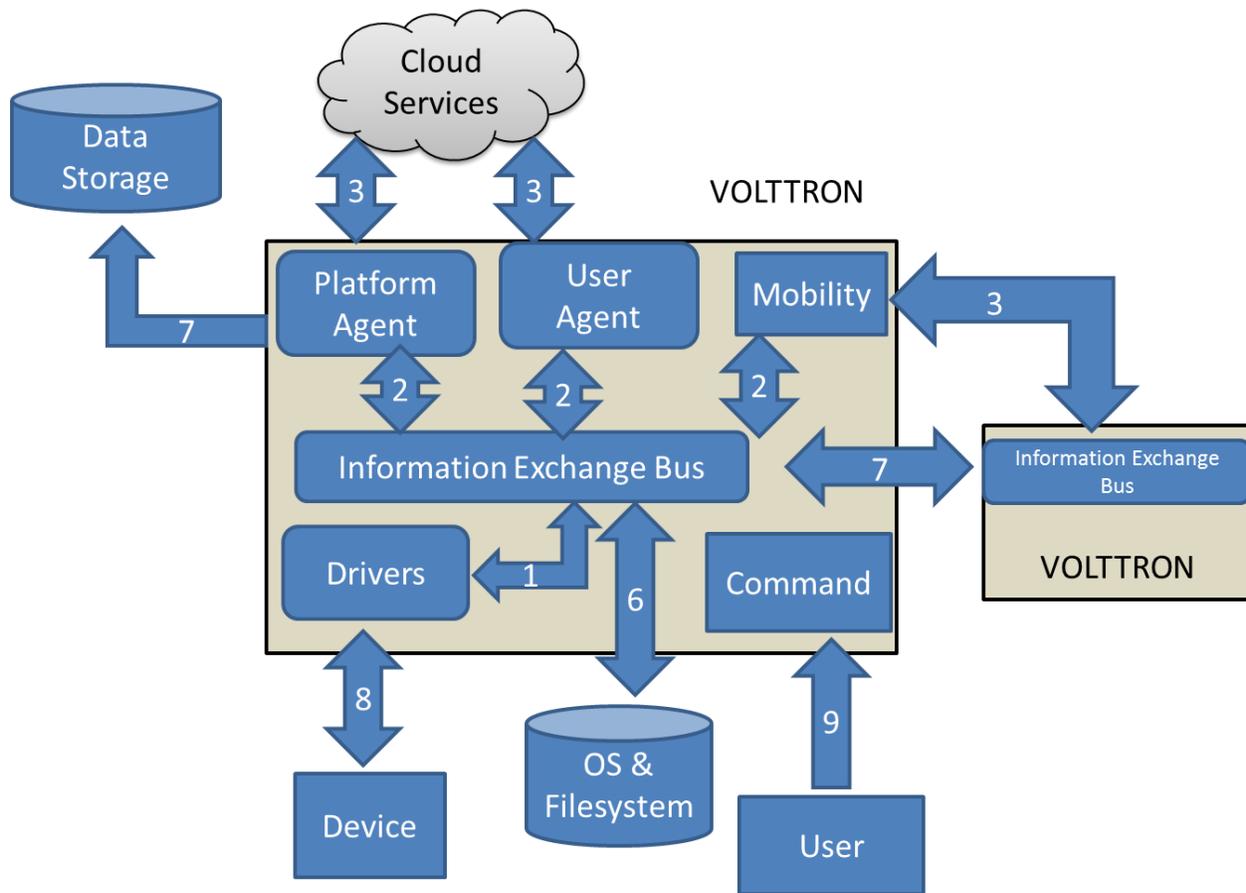


Figure 3.21. Schematic of VOLTRON™ Internal and External Components of the platform

This section discusses VOLTRON™ security and vulnerabilities. First, possible attack vectors against the VOLTRON™ software are indicated as well as associated risks and mitigations for reducing, alleviating, or even eliminating the risks. Some threats also include future strategies for improving the mitigations and even further reducing the risk. Each vulnerability follows the following template:

- Description of vulnerability/threat.
- R:** Associated risk indicating what might occur if the vulnerability is exercised.
- M:** Mitigation that should/will be implemented to reduce or eliminate the associated risk.
- F:** Future strategies for mitigation (optional).

3.6.1 Vulnerabilities Associated with Communications between VOLTTRON™ and Other Services (e.g. Cloud, etc.) (Interfaces 3 and 7 in Figure 2.21)

The VOLTTRON™ platform allows agents to act as proxies to external resources to move information between a service and the platform (3). In addition, the platform utilizes an external service for storing data collected from devices managed by the platform and data logged by applications (7). Vulnerabilities associated with these interfaces are listed below:

1. Communication between agents and the Cloud/external historian can be intercepted, tampered with, or snooped upon by a third party.

R: A remote entity can insert itself between VOLTTRON™ and external entities. Once in the path, the remote entity can tamper with communications, affecting integrity, or snoop the communications, affecting confidentiality.

M: VOLTTRON™ uses standardized communication protocols—transport layer security/secure socket layer) (TLS/SSL—when communicating with external entities. These protocols provide both message integrity and confidentiality services. Identities are authenticated by means of X.509 certificates.

2. Communication between agents and the Cloud/external historian can be stopped by either communications network availability or third-party DoS.

R: A third party can, by means of DoS attacks, or by other means, temporarily interrupt communication between VOLTTRON™ and external entities. This could result in loss of data or sub-optimal behavior of the control system.

M: VOLTTRON™ buffers all data that cannot be transmitted to external services. The buffer amount is only limited by the storage available to the platform.

M: VOLTTRON™ has the ability to go to safe control mode if it cannot communicate with an external controlling entity (that, for example, sets control policies). It is expected that the handling of loss of control policy communication between agents and devices should be handled by the agents. The VOLTTRON™ team will provide best practice guidance for agent developers.

3. An external entity communicating with VOLTTRON™ can be compromised and data coming from or going to VOLTTRON™ can be intercepted at the non-VOLTTRON™ end of the communication channel.

R: If an entity communicating with VOLTTRON™ is compromised, then data coming from or going to VOLTTRON™ can be intercepted or modified.

M: All entities that communicate with any control system should be protected as well as the control system itself. If the security of a third-party entity is under question, VOLTTRON™ developers recommend not trusting the entity at all. The VOLTTRON™ team will provide best practice guidance to protect control systems.

4. Communication between agents and the Cloud (or other outside, network-based) services could be used as an attack vector to compromise the system by intercepting unencrypted or improperly encrypted communications, compromising the remote system, or by performing other attacks.

R: A remote entity could send carefully crafted messages to agents that could cause the execution of unauthorized code.

M: Agent code will be reviewed for security issues and malicious intent. Initially (FY 2015 and FY 2016), PNNL will review contributions from external contributors before integrating the contributions into the core VOLTTRON™ code base. Eventually, this review function will be transferred to the organization that maintains VOLTTRON™. Agent developers should use encrypted and authenticated services when available and should validate all inputs from other sources before use. The archiver uses TLS/SSL when communicating with the archival service.

F: Full agent containerization will limit the effects and reach of a compromised agent.

5. Agents may communicate with any system in the Cloud with which any other agent is also able to communicate (assuming firewall allows such communication).

R: If an agent is inadequately authenticated or knows the credentials of the remote account, it could send malicious communications.

M: Agent code will be reviewed for security issues and malicious intent.

F: Agents will be sandboxed to disallow unauthorized communications.

R: An agent may exfiltrate data to systems in the Cloud.

M: Agent code will be reviewed for security issues and malicious intent.

M: Firewall rules may be used to limit communications with remote systems.

F: Agents will be sandboxed to disallow unauthorized communications.

3.6.2 Vulnerabilities Associated with Communications between Multiple VOLTTRON™ Instances (Interfaces 4 and 5 in Figure 2.21)

VOLTTRON™ platforms can communicate with each other through the use of the multi-node communication service (5). This service is an extension of the local message bus and forwards messages on a specific local topic to the message bus of another VOLTTRON™ instance. VOLTTRON™ also provides a Mobility Service (4), which enables agents to move between platforms and enables administrators to provision VOLTTRON™ devices in the field. Vulnerabilities associated with these interfaces are listed below:

1. Agents may communicate between the messaging buses of platforms located on different systems (if the MultiNode agent is enabled).

R: Agents may subscribe to any topic, without limit, when remote subscriptions are enabled on the remote platform.

M: Agent code will be reviewed for the proper use of remote subscriptions.

F: Topic filtering/authorization will allow limiting remote subscriptions.

R: Agents may publish to any topic, without limit, when remote publishing is enabled on the remote platform.

M: Agent code will be reviewed for the proper use of remote publishing.

F: Topic filtering/authorization will allow limiting remote publishing.

R: Multi-node messages may cross uncontrolled networks providing opportunity for interception or modification.

M: Multi-node messaging uses the elliptic-curve encryption technology of ZeroMQ's CurveZMQ protocol to authenticate and encrypt traffic between nodes (when a keyset is provided).

F: May consider replacing the use of CurveZMQ with secure shell (SSH) forwarding/tunneling in a future release.

R: Two additional TCP ports are required for multi-node communication and may be susceptible to DoS attacks. A third port must be opened when the Mobility Service is enabled.

M: Firewall rules may be applied to help limit the effectiveness of such attacks. VOLTRON™ provides no protection itself against DoS attacks.

F: Opening a single port and multiplexing all traffic will limit the number of network ports requiring exposure to the Internet.

2. Agents may move between platforms over the network introducing them to possible man-in-the-middle attacks, spoofing, or other network-directed attacks.

R: An unauthorized user might intercept an agent and modify it for malicious use or create their own agent and send it on behalf of a platform they are not authorized to use.

M: SSH tunnels are used to authenticate and encrypt communications between platforms. Agent code is cryptographically signed using X.509 certificates. SSH keys are managed using standard SSH configuration files and signing keys are managed using X.509 certificates.

F: SSH key usage will be converted to use the X.509 certificate infrastructure.

3. JSON-RPC (JavaScript object notation-remote procedure call) function calls are issued over an SSH tunnel to coordinate agents moving between platforms.

R: JSON messages must be completely read into memory. Receiving many extremely large messages could result in a low memory condition. JSON was chosen for serialization because of

its simplicity and its unlikeliness to be the source of inadvertent vulnerabilities.

F: A maximum message size could limit memory used. Limiting the number of concurrently parsed messages could help reduce memory usage, but at the cost of slowing communications.

3.6.3 Vulnerabilities Associated with Agent Multi-Tenancy inside a VOLTTRON™ Platform (Interfaces 1,2, and others in Figure 2.21)

VOLTTRON™ supports multiple agents and services running simultaneously. This ability is referred to as multi-tenancy. The following are potential vulnerabilities related to agent and service multi-tenancy inside the platform:

1. All agents run under the same user account with the same privilege level.

R: A malicious agent can interfere with other agents, possibly sending them signals, killing them, or overwriting data.

M: Agent code will be reviewed for security issues and malicious intent.

M: Agent code is signed (multiple times) and verified before each execution. This prevents an unauthorized third party from tampering with agent code.

F: Full containerization of agent code is planned for a future release and will effectively isolate agents.

2. Agent code runs under the same user account and at the same privilege level as the platform supervisory daemon.

R: A malicious agent can interfere with the platform supervisor, killing it and/or overwriting data. It could also assume the supervisor's role as manager of the communications bus, allowing it to intercept, modify, and/or drop agent communications. This also includes the ability to remove CPU and memory limits (restricted additions).

M: Agent code will be reviewed for security issues and malicious intent.

M: Agent code is signed (multiple times) and verified before each execution. This prevents an unauthorized third party from tampering with agent code.

F: Full containerization of agent code will effectively isolate and hide agents from the supervisor.

3. Local communication between agents over the message bus is unauthenticated.

R: A malicious agent may mimic other agents or the supervisor and send messages on their behalf, potentially causing the creation of unauthentic data or the unauthorized actuation of a device.

M: Agent code will be reviewed for security issues and malicious intent.

M: Agent code is signed (multiple times) and verified before each execution. This prevents an unauthorized third party from tampering with agent code. Platform will not allow execution of unauthorized agents.

M: Operating system and VOLTTRON™-level features defend against compromise of the internal messaging bus.

F: Authentication of inter-agent communications will be supported for validating message authenticity.

F: Messaging bus may be enhanced to support policy-based secure access to the messaging bus.

4. Local communication between agents over the message bus is unencrypted.

R: A malicious agent may subscribe to every message sent on the message bus and retransmit it or use it for other unintended purposes.

M: Agent code will be reviewed for security issues and malicious intent.

M: Agent code is signed (multiple times) and verified before each execution. This prevents an unauthorized third party from tampering with agent code. Platform will not allow execution of unauthorized agents.

M: Operating system and VOLTTRON™-level features defend against compromise of the internal messaging bus.

F: Encrypting the body of inter-agent communications will be supported.

3.6.4 Vulnerabilities Associated with Communicating with and Controlling Devices (Interface 8 in Figure 2.21)

VOLTTRON™ drivers (8) allow the platform to both collect data from devices and send control commands. These drivers utilize protocols such as MODBUS, BACnet, or custom-built software to communicate with the device and use the platform's message bus to communicate with agents on the platform. Overall security of the underlying communications protocol such as BACnet is outside the scope of this document. VOLTTRON™ developers recommend using appropriate cyber security measures to protect the underlying protocol. Refer to the Appendix A3.6.7Appendix A and the article by Neilson (2013)¹ for a positive example on how to secure a control systems network.

1. Unsecured communications between VOLTTRON™ and legacy control devices (e.g. Modbus, BACnet) may be intercepted and modified by a third party.

R: A third party can modify communications between VOLTTRON™ and controlled devices using legacy protocols. It is even possible to impersonate a controlled device. An agent can then react incorrectly to information coming from such a device.

M: Security measures as described in NIST SP800-82² and Neilson are to be used to protect legacy control system devices that do not have sufficient security protections built-in.

M: VOLTTRON™ agents can be written to validate information being received from legacy devices to check for range, historically known trends, etc.

¹ Neilson, C. 2013. *Securing a Control Systems Network*. ASHRAE Journal. November 2013.

² NIST SP800-82 Guide to Industrial Control Systems (ICS) Security accessed at http://csrc.nist.gov/publications/drafts/800-82r2/sp800_82_r2_draft.pdf on November 17, 2014.

M: VOLTTRON™ device drivers are written to prevent potential exploits from “overflow” type attacks to gain access to the platform by means of data being passed by a legacy device. The VOLTTRON™ platform cannot be compromised by means of incorrect data streams.

2. Agents may communicate directly with local devices bypassing the platform drivers.

R: An agent could send commands outside the supervision of the scheduler/actuator causing equipment to operate in an unsafe way.

M: Agent code will be reviewed for security issues and malicious intent before deployment.

M: Agent code is signed (multiple times) and verified before each execution. This prevents an unauthorized third party from tampering with agent code. Platform will not allow execution of unauthorized agents.

M: If underlying protocol supports it, VOLTTRON™ can implement secure and authenticated communications. For example, for thermostats that support SSL communication, VOLTTRON™ can communicate with the device using SSL.

F: Agents will be sandboxed to disallow unauthorized communications.

3. A device could be physically tampered with.

R: The device could be modified to send inaccurate readings in an attempt to make agents take incorrect and possibly damaging actions.

M: Building and property owners must maintain awareness of their devices to ensure protection and follow best practice guidance described by Neilson (2013) and the guidance provided in the Appendix of this report.

M: Building owners must implement appropriate physical security measures.

F: In an extension of the existing fault detection work, agents could be developed to monitor devices for improper and unexpected behavior.

3.6.5 Vulnerabilities Associated with the Underlying Operating System and File System (Interface 6 in Figure 2.21)

VOLTTRON™ is built on top of a modern Linux operating system. While the platform addresses threats associated with the platform and its interfaces, equal thought needs to be given to the security of the underlying operating system. The VOLTTRON™ team relies on comprehensive security hardening of the operating system as well as staying up-to-date with periodically applied patches to further enhance the security of the instances deployed in the field. The following vulnerabilities (not an exhaustive list) are related to the interface between VOLTTRON™ and the underlying operating system:

1. Underlying operating system platform can be vulnerable to attacks because of incorrect or incomplete security update patching.

R: If a platform is not kept up-to-date with respect to security patches, an attacker will be able to compromise the system and gain access.

M: All VOLTTRON™ systems deployed in the field are configured to automatically download and apply security patches. VOLTTRON™ developers recognize that in some environments, unattended upgrades are not practical or even can be dangerous. In this case, it is recommended that an engineer apply security updates on a weekly or monthly basis.

M: VOLTTRON™ systems are protected by appropriate network security measures such as host-based firewalls, intrusion detection, and security monitoring tools to prevent unauthorized access.

2. Underlying operating system platform is insufficiently hardened.

R: Security settings of the underlying operating system are not managed correctly and allow overly broad ("loose") access. An example could be the "guest" account, or running unnecessary services.

M: All VOLTTRON™ platforms deployed in the field by VOLTTRON™ developers go through a security hardening process that includes turning off unnecessary services, restricting access to necessary services, activating host-based firewall controls, and allowing access to the system only by authorized users and hosts.

F: VOLTTRON™ team will document platform-hardening settings as part of the VOLTTRON™ user's guide.

3. Physical access to the VOLTTRON™ platform is not controlled.

R: An attacker that has physical access can circumvent many security measures implemented in software.

M: Physical access to any control system must be controlled. There are no exceptions.

M: Linux supports hard disk or volume encryption but this is not sufficient to defend against an attacker that has physical access.

4. Platform supervisor and agents run on a shared (multi-user) system (6).

R: Other users on the system might be able interact with supervisor or agent processes, files, and/or sockets. A malicious agent might be able to access processes, files, and sockets belonging to other users of the system.

M: In a production environment, the supervisor should run under its own unprivileged account. Files are writable only by the supervisor process's owner. Sensitive files are only readable by the supervisor process's owner. Appropriate file system permissions are set on Unix domain sockets to prevent their use by unauthorized users and/or peers are authenticated. TCP/UDP sockets require authentication.

F: Full containerization of agent code will effectively isolate agents and hide much of the system from them.

R: An agent may consume too much memory, intentionally or through a programming error, causing the system to become unresponsive and forcing other applications and/or agents to terminate.

M: A resource monitor is used to place the agent in a memory Linux control group (cgroup) to limit memory usage to what was negotiated before execution.

F: An agent's out-of-memory (OOM) killer priority can be set lower than critical applications and higher-priority agents so that an OOM condition will cause lower priority agents to be killed first (restricted additions).

R: An agent may consume too many CPU cycles, intentionally or through a programming error, causing the system to become unresponsive and forcing other applications and/or agents to terminate.

M: A resource monitor is used to place the agent in a CPU cgroup to limit its CPU utilization to what was negotiated before execution (restricted additions).

Note: Appendix B has detailed recommendations for Linux Platform Hardening for VOLTRON™ users.

3.6.6 Vulnerabilities associated with user administration of the platform, user interfaces, etc. (Interface 9 in Figure 3.21)

VOLTRON™ platforms support an easy-to-use, command-line user interface for platform administration. As part of future development, based on user requests, the VOLTRON™ team will also develop a simple web management console that runs on the platform as well as a comprehensive, web-based management system titled VOLTRON™ Central. The vulnerability list discussed below is limited to discussion of the command-line interface only. This document will be updated during the implementation of the VOLTRON™ 3.0 features.

1. The platform is locally controlled via a Unix domain socket (9).

R: Any user with local access to the system has the potential to send command and control messages to the platform.

M: Access to the control socket is limited by file system permissions on the socket and the owner and group of processes attempting to connect to the socket are validated against an access control list in the platform configuration. The superuser may also be denied access.

2. TLS/SSL socket communications, RSA encryption, and X.509 certificate management and verification make use of the locally installed OpenSSL library.

R: Any vulnerability in OpenSSL, such as HeartBleed, could negatively affect the security of the platform and potentially the system.

M: The system administrator or owner must keep the system up-to-date with the latest security patches, especially with regard to the OpenSSL libraries.

3. VOLTRON™ is written in Python and runs via the local Python installation.

R: Any vulnerability in Python could negatively affect the security of the platform and potentially the system.

M: The system administrator or owner must keep the system up-to-date with the latest security patches, especially with regard to the base Python installation.

4. VOLTTRON™ uses third-party Python libraries/packages.

R: A vulnerability in the platform's third-party dependencies could allow VOLTTRON™ to be compromised.

M: The VOLTTRON™ developers use mature and actively developed third-party packages with good reputations for stability and security; however, any complex code is likely to suffer bugs that may lead to compromise. While code written in Python is much less susceptible to certain attacks, it is recommended that third-party libraries are regularly updated to the latest compatible version to take advantage of security patches.

3.6.7 Summary of VOLTTRON™ Security Features

VOLTTRON™ security features are based on the mitigations discussed in the previous section (denoted by M). These security features are summarized below:

- VOLTTRON™ is built on Linux to take advantage of its many built-in security features, such as powerful file system permissions, user management, Linux capabilities configuration, control groups, and a first-class firewall.
- VOLTTRON™ accesses remote resources as securely as possible, utilizing the highest version of TLS/SSL protocols and with the largest key size available to both endpoints. Within VOLTTRON™, OpenSSL is used for TLS/SSL encrypted links. The system's OpenSSL libraries are kept as up-to-date as possible to prevent vulnerabilities such as HeartBleed.
- For multi-platform communication, VOLTTRON™ uses remote ZeroMQ sockets using CurveZMQ elliptical curve encryption. Keys must be configured for links to be encrypted.
- Paramiko SSH Python package is used to provide RSA-encrypted communications between platforms for agent mobility. Most key sizes and encryption scheme defined in version 2 of the SSH protocol (SSHv2) are supported.
- Linux control groups (cgroups) CPU and memory subsystems are used to limit excessive processor and memory usage.
- Platform control (Unix domain) socket utilizes a mixture of file permissions and access control lists to limit access to authorized users.
- Code is peer reviewed for correctness and security.
- Agent code and packages are signed and verified using RSA encryption with X.509 certificates. Unsigned code is not executed unless explicitly allowed by the administrator.

Appendix A: An Example Best Practice for Securing Building Control Networks

An Example Best Practice for Securing Building Control Networks

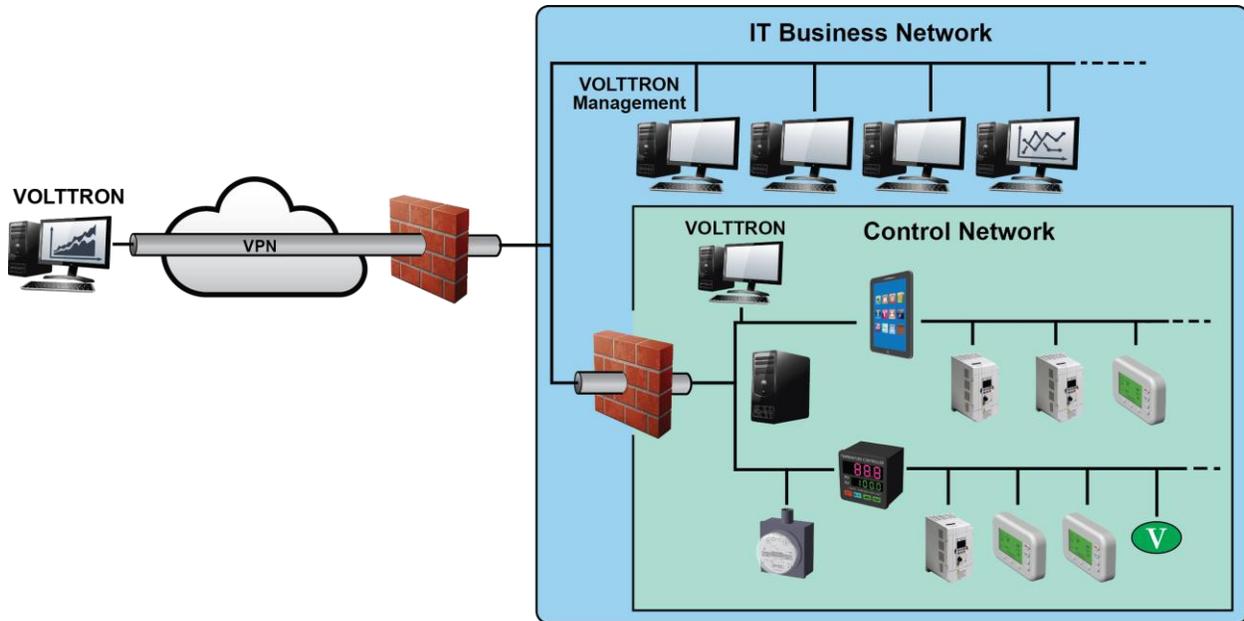


Figure A.1. Secure Building Control Network

Figure A.1 shows an example best practice of how to secure building control networks. In this example, the control network is completely segregated with its own network and a firewall. The control network is not connected to either the organizational information technology (IT) network or to the Internet directly. This type of segregation reduces the attack points for the building control network. The building control network is protected from the organizational IT network. The access to the control network from the IT network can be tailored by setting up the required firewall rules. Although the graphic above shows two independent networks, this type of security can also be achieved using virtual local area networks. For more details on this guidance, please refer to Neilson (2013)¹. Also, there is an instance of VOLTRON™ running on the segregated building control network and an instance of VOLTRON™ running on the IT network. These two instances of VOLTRON™ can communicate securely using security features built into the VOLTRON™ platform.

¹ Neilson, C. 2013. *Securing a Control Systems Network*. ASHRAE Journal. November 2013.

Appendix B: **Linux Platform Hardening for VOLTTRON™**

Linux Platform Hardening for VOLTTRON™

The current version of this text can be retrieved from:

<https://github.com/VOLTTRON/volttron/wiki/Linux-Platform-Hardening-Recommendations-for-VOLTTRON-users>.

B.1 Introduction

VOLTTRON™ is an agent-based application development platform for distributed control systems. VOLTTRON™ itself is built with modern security principles in mind [security-wp] and implements many security features for hosted agents. However, VOLTTRON™ is built on top of Linux, and the underlying Linux platform also needs to be secured in order to declare the resulting control system as "secure." Any system is only as secure as its weakest link. The rest of this note is dedicated to making recommendations for hardening of the underlying Linux platform that VOLTTRON™ uses. No system can be 100% secure, thus the cyber security strategy that is recommended in this document is based on risk management.

B.2 Linux System Hardening

The following are the non-exhaustive recommendations for Linux hardening from the VOLTTRON™ team:

- **Physical Security:** Keep the system in locked cabinets or a locked room. Limit physical access to systems and to the networks to which they are attached. The goal should be to avoid physical access by untrusted personnel. This could be extended to blocking or locking USB ports, removable media drives, etc. Drive encryption could be used to avoid access via alternate-media booting (off USB stick or DVD) if physical access cannot be guaranteed. The downside of drive encryption would need to enter a passphrase to start system. Alternately, the Trusted Platform Module (may be used, but the drive might still be accessible to those with physical access. Enable chassis intrusion detection and reporting if supported. If available, use a physical tamper seal along with or in place of an interior switch.
- **Low level device Security:** Keep firmware of all devices (including BIOS) up-to-date. Password-protect the BIOS. Disable unneeded/unnecessary devices including serial, parallel, USB, Firewire, etc. ports; optical drives; wireless devices, such as Wi-Fi and Bluetooth. Leaving a USB port enabled may be helpful if a breach occurs to allow saving forensic data to an external drive.
- **Boot security:** Disable automounting of external devices. Restrict the boot device. Disable PXE and other network boot options (unless that is the primary boot method). Disable booting from USB and other removable drives. Secure the boot loader. Require an administrator password to do anything but start the default kernel. Do not allow editing of kernel parameters. Disable, remove, or password-protect emergency/recovery boot entries.
- **Security Updates:** Configure the system to automatically download security updates. Most security updates can be installed without rebooting the system, but some updates (e.g. shared libraries, kernel, etc.) require the system to be rebooted. If possible, configure the system to install the security updates automatically and reboot at a particular time. We also recommend reserving the reboot time (e.g.

1:30AM on a Saturday morning) using the Actuator Agent so that no control actions can happen during that time.

- System Access only via Secured Protocols: Disallow all clear text access to VOLTRON™ systems. No telnet, no rsh, no ftp and no exceptions. Use SSH to gain console access, and scp/sftp to get files in and out of the system. Disconnect excessively idle SSH Sessions.
- Disable remote login for "root" users. Do not allow a user to directly access the system as the "root" user from a remote network location. Root access to privileged operations can be accomplished using "sudo." This adds an extra level of security by restricting access to privileged operations and tracking those operations through the system log.
- Manage users and usernames. Limit the number of user accounts. Use complex usernames rather than first names.
- Authentication. If possible, use two-factor authentication to allow access to the system. Informally, two-factor authentication uses a combination of "something you know" and "something you have" to allow access to the system. RSA SecurID tokens are commonly used for two-factor authentication but other tools are available. When not using two-factor authentication, use strong passwords and do not share accounts.
- Scan for weak passwords. Use password cracking tools such as John the Ripper (<http://www.openwall.com/john/>) or nmap with password cracking modules (<http://nmap.org>) to look for weak passwords.
- Use Pluggable Authentication Modules (PAM) to strengthen passwords and the login process. We recommend:
 - pam_abl: Automated blacklisting on repeated failed authentication attempts
 - pam_captcha: A visual text-based CAPTCHA challenge module for PAM
 - pam_passwdqc: A password strength checking module for PAM-aware password changing programs
 - pam_cracklib: PAM module to check the password against dictionary words
 - pam_pwhistory: PAM module to remember last passwords
- Disable unwanted services. Most desktop and server Linux distributions come with many unnecessary services enabled. Disable all unnecessary services. Refer to your distribution's documentation to discover how to check and disable these services.
- Just as scanning for weak passwords is a step to more secure systems, conduct regular network scans using nmap (www.nmap.org) to find what network services are being offered. Use nmap or similar tools very carefully on BACnet and modbus environments. These scanning tools are known to crash/reset BACnet and modbus devices.
- Control incoming and outgoing network traffic. Use the built-in host-based firewall to control who/what can connect to this system. Many IP tables frontends offer a set of predefined rules that provide a default deny policy for incoming connections and provide rules to prevent or limit other well-known attacks (i.e., rules that limit certain responses that might amplify a DoS attack). UFW (uncomplicated firewall) is a good example. For example, if the system administrators for the VOLTRON™ device are all located in 10.10.10.0/24 subnetwork, then allow SSH and SCP logins

from only that IP address range. If VOLTTRON™ system exports data to a historian at 10.20.20.1 using TCP port 443, allow outgoing traffic to that port on that server. The idea here is to limit the attack surface of the system. The smaller the surface, the better we can analyze the communication patterns of the system and detect anomalies. While some system administrators disable network-based diagnostic tools such as ICMP ECHO responses, The VOLTTRON™ team believes that this hampers usability. As an example, monitoring which incoming and outgoing firewall rules are triggering can be accomplished with this command: `watch --interval=5 'iptables -nvL | grep -v "0 0"' .`

- Rate-limit incoming connections to discourage brute-force hacking attempts. Use a tool such as fail2ban (http://www.fail2ban.org/wiki/index.php/Main_Page) to dynamically manage firewall rules to rate-limit incoming connections and discourage brute-force hacking attempts. Additionally, sshguard (<http://www.sshguard.net/>) is similar to fail2ban but only used for SSH connections. Further rate limiting can be accomplished at the firewall level. As an example, you can restrict the number of connections used by a single IP address to your server using iptables. Only allow 4 SSH connections per client system:

```
iptables -A INPUT -p tcp --syn --dport 22 -m connlimit --
connlimit-above 4 -j DROP
```

You can limit the number of connections per minute. The following example will drop incoming connections if an IP address makes more than 10 connection attempts to port 22 within 60 seconds:

```
iptables -A INPUT -p tcp -dport 22 -i eth0 -m state --state NEW -
m recent --set
```

```
iptables -A INPUT -p tcp -dport 22 -i eth0 -m state --state NEW -
m recent\ --update --seconds 60 --hitcount 10 -j DROP
```

- Use a file system integrity tool to monitor for unexpected file changes. Tools such as tripwire (<http://sourceforge.net/projects/tripwire/>) to monitor filesystem for changed files. Another file integrity checking tool to consider is Advanced Intrusion Detect Environment (<http://aide.sourceforge.net/>).
- Use filesystem scanning tools periodically to check for exploits. Available tools such as checkrootkit (<http://www.chkrootkit.org>), rkhunter (<http://rkhunter.sourceforge.net>) and others should be used to check for known exploits on a periodic basis and report their results.
- VOLTTRON™ does not use Apache or Require It. If Apache is being used, we recommend using mod_security and mod_evasive modules.

B.3 System Monitoring

- Monitor system state and resources. Use a monitoring tool such as Xymon (<http://xymon.sourceforge.net>) or big brother (<http://www.bb4.org/features.html>) to remotely monitor the system resources and state. Set the monitoring tools to alert the system administrators if anomalous use of resources (e.g. connections, memory, etc.) is detected. An administrator can also use Unix commands such as netstat to look for open connections periodically.

- Watch system logs and get logs off the system. Use a utility such as logwatch (<http://sourceforge.net/projects/logwatch/files/>) or logcheck (<http://logcheck.org>) to get a daily summary of system activity via email. For Linux distributions that use systemd, use journalwatch (<http://git.the-compiler.org/journalwatch/>) to accomplish the same task. Additionally, use a remote syslog server to collect logs from all VOLTTRON™ systems in the field at a centralized location for analysis. A tool such as splunk is ideal for this task and comes with many built-in analysis applications. Another benefit of sending logs remotely off the platform is the ability to inspect the logs even when the platform may be compromised.
- An active intrusion sensor such as PSAD (<http://cipherdyne.org/psad/>) can be used to look for intrusions as well.

B.4 Security Testing

Every security control discussed in the previous sections must be tested to determine correct operation and impact. For example, if we inserted a firewall rule to ban connections from an IP address such as 10.10.10.2, then we need to test that the connections actually fail. In addition to functional correctness testing, common security testing tools such as Nessus (<http://www.tenable.com/products/nessus>) and nmap (<http://nmap.org>) should be used to perform cyber security testing.

B.5 Conclusion

No system is 100% secure unless it is disconnected from the network and is in a physically secure location. VOLTTRON™ team recommends a risk-based cyber security approach that considers each risk, and the impact of an exploit. Mitigating technologies can then be used to mitigate the most impactful risks first. VOLTTRON™ is built with security in mind from the ground up. But it is only as secure as the operating system of the host machine. This document is intended to help VOLTTRON™ users to secure the underlying Linux operating system to further improve the robustness of the VOLTTRON™ platform.



Pacific Northwest
NATIONAL LABORATORY

*Proudly Operated by **Battelle** Since 1965*

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

U.S. DEPARTMENT OF
ENERGY

www.pnnl.gov