DIMS Architecture Design Documentation

Release 2.10.0

David Dittrich, Stuart Maclean

Aug 16, 2017

Contents

CHAPTER 1

Scope

Identification

This Architecture Design document (version 2.10.0) describes the proposed high-level design of the Distributed Incident Management System (DIMS) architecture. Its purpose is to provide the reader with an overview of the major system components of the former Public Regional Information Security Event Management (PRISEM) system, the Operational Security Trust (Ops-Trust) portal system (now *Trident*), and the components of the DIMS *dashboard* front-end and data processing back end that integrate these two existing systems.

DIMS is funded by the Department of Homeland Security under contract HSHQDC- 13-C-B0013. For more information, see the documents DIMS Operational Concept Description v 2.9.0, DIMS System Requirements v 2.9.0, and other documents referenced in Section *Referenced documents*.

The scope of this document is limited to description of the architectural elements, data types (and their sources, volumes, and retention periods), data flows, user interfaces, etc. It is assumed the reader is familiar with the underlying motivations for the system as described in the DIMS Operational Concept Description v 2.9.0 and DIMS System Requirements v 2.9.0 documents.

Attention: This document was originally written in the months after the initial start of period of performance of the DIMS project contract, which was September 13, 2013. Things have changed over the Base Period, Option Period, and two extensions. This, and all related DIMS documents, were originally written as *forward-looking* documents using future tense. This, and all related project documents, are undergoing updates to reflect changes that have occured (including switching to past tense and renaming as necessary) on a best-effort basis. Keep this in mind while reading this document, and feel free to report any errors you encounter by filing a bug report or issuing a pull request.

Document overview

The structure of this document has been adapted principally from MIL-STD-498 (see Section *Referenced documents*). Following this section are:

- Section *Referenced documents* lists related documents.
- Section *System-wide design decisions* describes the system-wide decisions that guide the design of the Distributed Incident Management System.
- Section *DIMS architectural design* describes the DIMS architectural design.
- Section *DIMS detailed design* provides details on the hardware and software subsystem design.
- Section *Requirements traceability* describes traceability back to requirements expressed in the DIMS System Requirements v 2.9.0 document.
- Section *Notes* provides an alphabetical listing of acronyms and abbreviations used in this document.
- Section *License* includes the copyright and software license under which DIMS is being released.

CHAPTER 2

Referenced documents

1. DIMS System Requirements v 2.9.0
2. DIMS Operational Concept Description v 2.9.0
3. dimsdockerfiles:usingdockerindims
4. DIMS Test Plan v 2.9.1
5. HSHQDC-13-C-B0013, "From Local to Gobal Awareness: A Distributed Incident Management System," Draft contract, Section C - Statement of Work (marked up version)
6. MIL-STD-498, Military Standard Software Development and Documentation, AMSC No. N7069, Dec. 1994.
7. D. Dittrich. PRISEM Analyst's Handbook, December 2013.
8. D. Dittrich. PRISEM System Administration Handbook, December 2013.
9. W. Gragido. Understanding Indicators of Compromise (IOC) Part I, October 2012. http://blogs.rsa.com/will-gragido/understanding-indicators-of-compromise-ioc-part-i/
10. M. Hamilton and D. Dittrich. An overview of the Public Regional Information Security Event Management Project, December 2013.
11. E. Hutchins, M. Cloppert, and R. Amin. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. In 6th Annual International Conference on Information Warfare and Security. Lockheed Martin Corporation, December 2011. http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf
12. H. Khurana, J. Basney, M. Bakht, M. Freemon, V. Welch, and R. Butler. Palantir: A Framework for Collaborative Incident Response and Investigation. In IDtrust '09: Proceedings of the 8th Symposium on Identity and Trust on the Internet, pages 38–51, New York, NY, USA, April 2009. ACM. http://middleware.internet2.edu/idtrust/2009/papers/05-khurana-palantir.pdf
13. R. S. C. Ieong. FORZA - Digital forensics investigation framework that incorporate legal issues. Digital Investigation, 3(Supplement-1):29–36, 2006. http://www.dfrws.org/2006/proceedings/4-Ieong.pdf
14. Mandiant. Using Indicators of Compromise to Find Evil and Fight Crime, August 2011. http://www.us-cert.gov/GFIRST/presentations/2011/Using_Indicators_of_Compromise.pdf

15. The Mitre Corporation. Standarizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX), 2012. http://makingsecuritymeasurable.mitre.org/docs/STIX-Whitepaper.pdf

CHAPTER 3

System-wide design decisions

Background on Existing Core Components

To understand what the DIMS system is intended to provide, it is important to understand its role in the context of distributed and collaborative incident response. DIMS leverages the capabilities of several existing systems each provide key functions necessary for incident response, but are not presently designed to work together. Integrating these capabilities will result in an increase in the capacity to respond.

Figure *Overview of DIMS System* depicts a high-level diagram of the dataflows between DIMS and related system.

DIMS provides a user interface layer on the front end, as well as a data processing layer on the back end, that integrates with several existing systems.

- The first is the Security Information Event Management (SIEM) system at the core of the PRISEM project, and the technologies associated with it to perform behavioral detection of malicious activity from network flow data and support forensic analysis of historic data to respond and recover from attacks that evade detective mechanisms. This system collects and processes tens of millions of security related events (and network flow records, if desired) and supports a collective approach to responding and recovering from security events.
- The second system is the Ops-Trust portal system, used by a community of several hundred computer security professionals with operational and research roles in industry, government, and academia. This system is primarily designed to facilitate trust group maintenance and communication to deal with emerging threats and events of international scope. (It is now in its second incarnation, as the *Trident* system).
- The third are the suite of "big data" style open source unstructured data storage, log processing, log visualization, and other tools that are part of the ELK stack, MozDef, and CIF.
- Additional tools that can be used for visualization can be similarly integrated (such as Mal4s), by building them into the system deployment infrastructure like any other components used in DIMS. This type of *framework* model, if generalized, allows any of a number of open source security tools to be made available to the incident responder.

The DIMS software system will bring these systems together into a collaborative environment for shared analysis and shared response of shared threats, both within a regional trust community, as well as across multiple such trust communities in other regions. Through vertical sharing of indicators of compromise from US-CERT to the regional level, and lateral sharing across regional entities, the objective is to scale actionable information sharing to state, local,

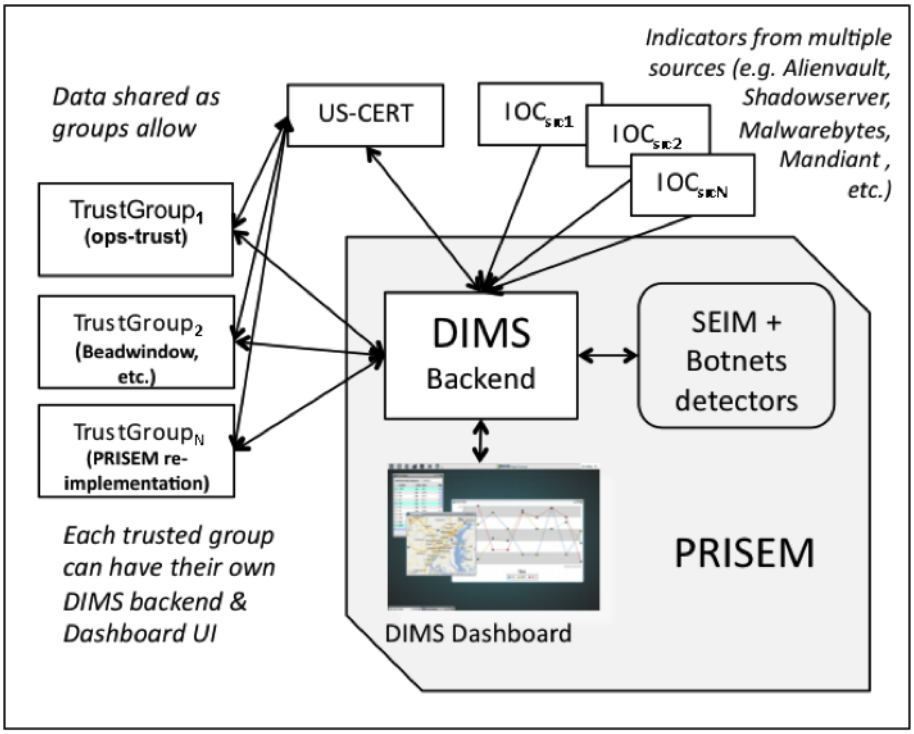


Fig. 3.1: Overview of DIMS System

territorial, and tribal (SLTT) government entities across the United States, and extend the sharing to international trust groups who make up the global fabric of the internet.

Figure *Data Flows Between Stakeholders* depicts the data flows between a subset of the stakeholders who will be using the DIMS software system. The solid lines depict data that has the highest degree of sensitivity and trust, often being transmitted in un-redacted form (possibly tagged with TLP indicators for most restricted sharing). The dashed lines depict data flows that are at lower levels of trust, and may be transmitted only in redacted form (possibly tagged with TLP indicators for the least restricted sharing). The type of data shared may be structured IOC and Observables in STIX format, Course of Action information in either PDF or structured format, *Situational Awareness Report* (*SITREP*) documents that describe observed campaign level activity at a high level, possibly with structure data containing IOCs or Observables to assist recipients in searching for related activity, and incident reports that may similarly be a combination of human-readable PDF and machine-readable IOCs/Observables. There are two types of data that will be shared in most use cases: high-frequency, high-volume, automated data feeds of *reputation* data and IOCs/Observables coming from analytic and research groups; low-frequency, low-volume, manually triggered bundles of IOCs/Observables, Course of Action information, and/or high-level SITREPs for specific incident-level up to campaign-level activity.

The DIMS software, layered on top of the Trident portal system as illustrated in Figure *DIMS and Trident Component Stack*, will facilitate production of these reports and transmission/reception of structure data files and facilitate automated processing of the structure data files to pre-process data for an analyst to consume when ready, rather than forcing the analyst to do a lot of work manipulating files, processing their contents, and manually entering data into report generation front ends in web based portals. (See also Figure *DIMS and Trident Component Interfaces*.)

Figure *PRISEM Initial Deployment and Flows* depicts the high-level data flow relationships for the Security Information Event Management (SIEM) system and Botnets detector subsystem used in the PRISEM project as it was initially deployed in 2009. The City of Seattle (the first and to this date largest participant organization) has multiple security devices sending event logs into the system. It also generates NetFlow V5 records that are processed by real-time detectors, and archived for historical query capability. The logs are collected one site, then forwarded to the central SIEM for processing at the University of Washington.

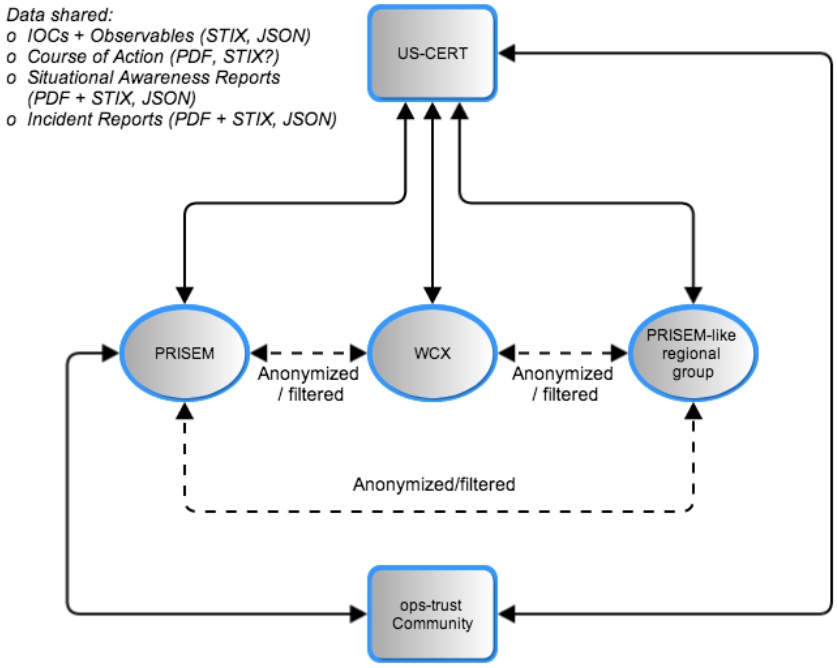


Fig. 3.2: Data Flows Between Stakeholders

Figure *Netflow Architecture* depicts a prototypical NetFlow collection and archiving model. The PRISEM system uses a slightly modified version of this model. Unlike the diagram in Figure 4, the PRISEM system processes NetFlow records as they enter the *NetFlow Collector* in the center of the diagram, sending copies to the Botnets system detectors. One of the processes receiving these records performs the storage task, however it converts the NetFlow V5 records to SiLK format before storing them. The SiLK tool suite is then used to process these historic logs (e.g., performing historic queries).

Figure *Botnets System High-Level Architecture* shows the high-level architecture of the Botnets network flow-based behavioral detector system. One or more NetFlow V5 feeds are combined into a single feed, which duplicates each NetFlow record and fans them out in to N different detectors. Each detector maintains its own state and sends out alerts when appropriate via SNMP, standard output to users in realtime, or to the Unix syslog service. (In Figure 5, syslog events are sent to a remote syslog server and processed by ZenOSS, an open source IT monitoring system. In the PRISEM system, all detectors alert via syslog, which are processed by the Log Matrix Threat Center application.)

Figure *PRISEM Architecture* shows the central system architecture of the PRISEM system. Shown in green are the Security Information Event Management (SIEM) system and event log archive in the bottom right. The box in the upper right depicts an instance of the network flow monitoring ("Botnets" detector system) and SiLK data archiving, which is typically housed on-site at participant networks due to sensitivity of network flow data. A central instance of the Collective Intelligence Framework (CIF) v0.1 database provides historic information about known malicious activity, which is used to pull watchlists that the Botnets detectors use for behavioral detection. A virtual machine server provides processing and AMQP broker functions to integrate data from multiple sources and correlate it across participating organizations, and optionally anonymize or filter any data prior to sharing. At present, a vendor-proprietary portal provides the graphical user interface front-end for participants, with the primary PRISEM systems residing behind a vendor-supported firewall, with command line utilities and AMQP access provided via an OpenVPN server for secure access. The DIMS dashboard will front-end this portal and support additional capabilities that are available on the PRISEM back-end via the AMQP broker.

Figure *Ops-Trust Architecture Diagram* shows the basic architecture of the Ops-Trust portal system. This system is a

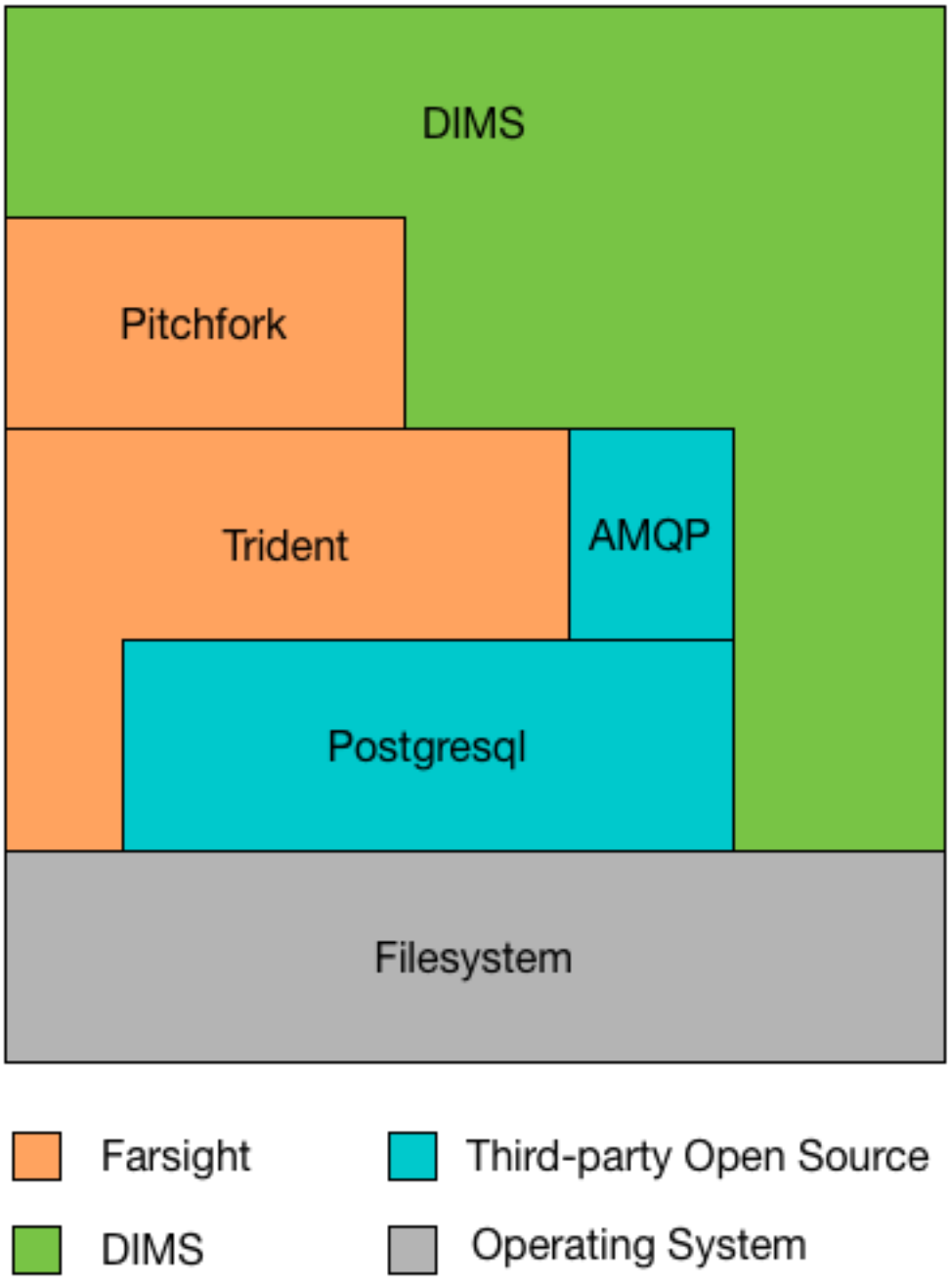


Fig. 3.3: DIMS and Trident Component Stack

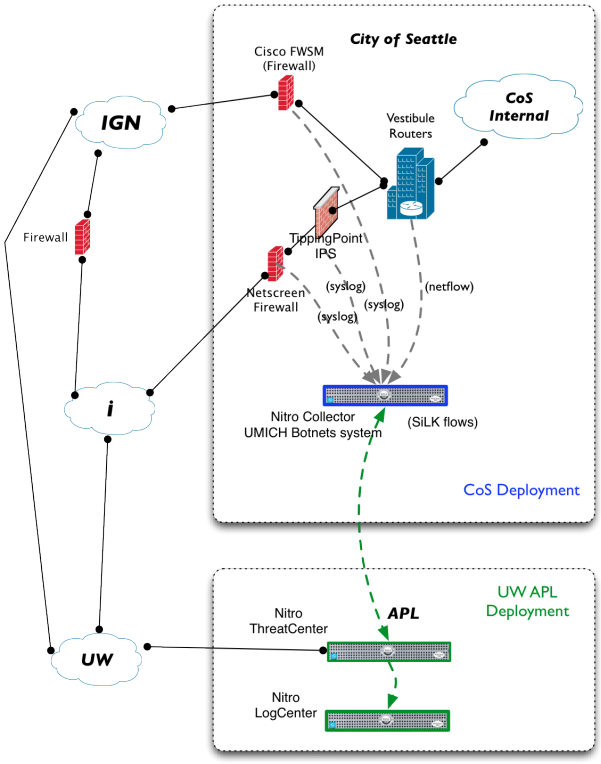


Fig. 3.4: PRISEM Initial Deployment and Flows

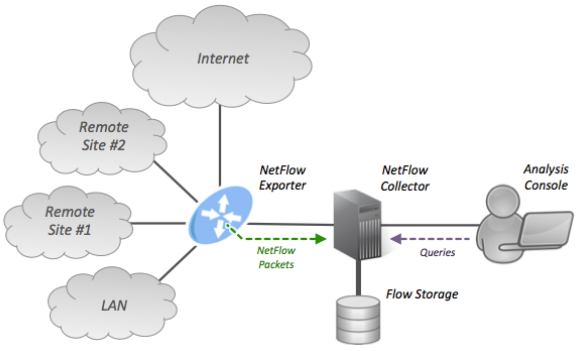


Fig. 3.5: Netflow Architecture

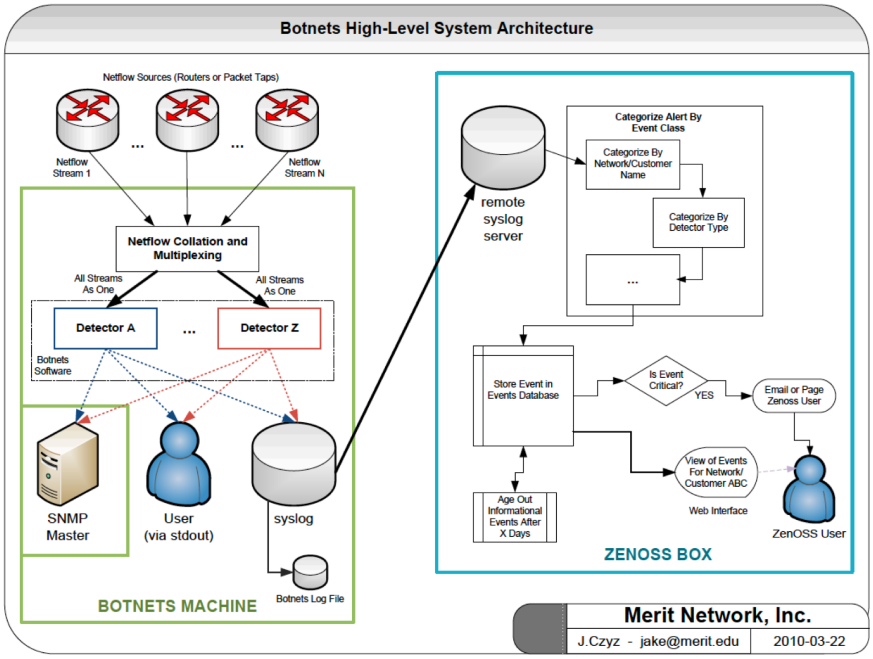


Fig. 3.6: Botnets System High-Level Architecture

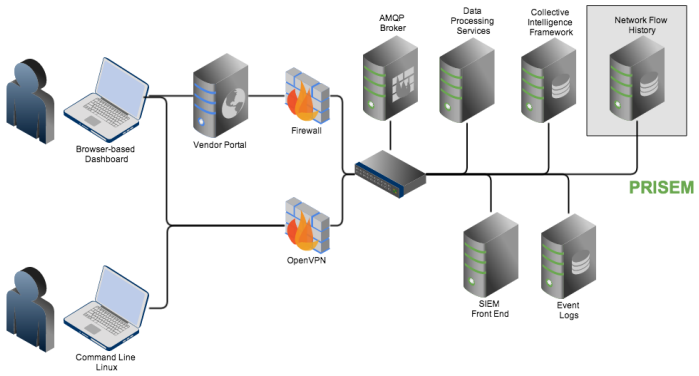


Fig. 3.7: PRISEM Architecture

combination of a web-based portal, a wiki for information archiving, an email server, and DNS and LDAP services tied to OpenID authentication services to provide single-signon capability. All of these services are provided via four separate virtual machines, co-resident in a single 1U server that is backed up off-site. The instance depicted in *Ops-Trust Architecture Diagram* is hosted on Ops-Trust hardware. A development instance was set up at the UW for DIMS development.

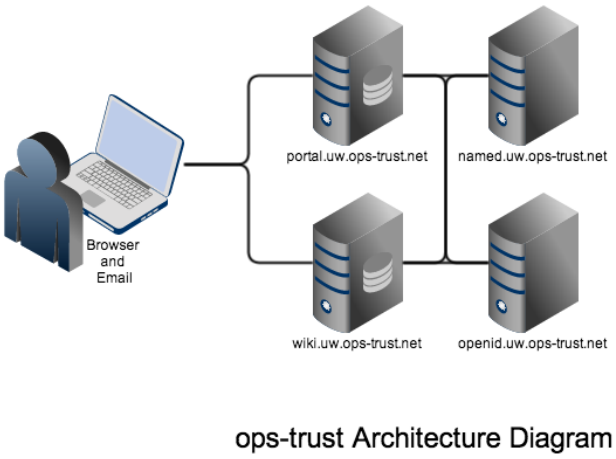


Fig. 3.8: Ops-Trust Architecture Diagram

The Ops-Trust portal stores attributes about each member. Figure *Ops-Trust Member Information Page* shows the account for the PI, which includes: user UUID; home time zone; nearest airport (to facilitate contact and meet-ups when one is on travel); how to contact via email, postal mail, SMS, IM, and phone; and current PGP encryption key. The portal lets you sign up for email lists, and switch between "trust groups". After signing up for (and optionally being approved for membership) email lists, the user is included on list email routed through the mail server, and granted access to the appropriate section of the wiki.

Fig. 3.9: Ops-Trust Member Information Page

The DIMS system will take advantage of the foundation of services provide by this portal in several ways. It will use it as a means of storing more information about users, the network assets they protect, the policies and mechanisms for anonymizing and filtering data based on TLP tagging, etc. It will also use it as a mechanism to distribute data to users as needed (e.g., alerts about email threads that pertain to the network assets they protect, providing a means to download OpenVPN certificates and SSH keys, as a mechanism for storing and organizing data associated with incidents and campaigns they are dealing with, etc.) The ability to manage encrypted communications and multiple email lists facilitates trusted communication and offers a basis for sending structured threat information in encrypted form, directly from one user to another, or from a user to all members of a list.

Software Development Methodology

As the DIMS system relies upon and integrates multiple existing open source software components, and code developed by the DIMS developers, the system is being developed using an Agile programming development methodology (as opposed to the classic *waterfall* development methodology with its sequential processes.) This document, therefore, is a *living document* that will be updated as the project proceeds and as cyclic input/feedback from users and testers is received. Sections to be addressed in future releases of this document are listed as TBA.

The DIMS project involves coordination of team members in multiple locations, multiple time zones, and multiple overlapping areas of responsibility. In order to communicate, coordinate, maintain momentum of project development, and meet deliverable requirements of the contract with the sponsor, all DIMS team members must be able to work asynchronously, independently, and be responsible for following task prioritization or asking for direction as necessary.

Use of Agile Development Methodology

Integration of existing open source tools requires research into how the existing tool is designed and how it functions, understanding how it processes inputs and outputs, and how it is configured.

The Agile methodology and Scrum methodology involve making small incremental changes based on simple user stories (short descriptions of what a user wants or needs), and making these changes on a short time frame (within a *sprint*, which is usually on the order of one or two weeks. (See [[agileDevelopment]] Agile development.)

Tasks are prioritized using the Jira Agile ticketing system, with the objective of completion of tasking within a 2-week sprint cycle. Weekly meetings are used to manage sprints.

Both source code, and system configuration files and installation instructions, are maintained using the Git source code control system using git-flow and hub, for eventual open source release on GitHub. This supports use of the Vincent Dreisen branching workflow to allow independent and isolated changes to be made, which are then to be tested prior to integration into more mainstream `develop` or `master` branches for release.

Use of Continuous Integration

The concepts of Continuous Integration and DevOps (also known as *agile system administration* or *agile operations*) for rapid development, testing, and release of a functional system are employed in order to build the overall system one component at a time, in a manner that can support the requirements specified in Adaptation requirements and [[continuousIntegration]] Continuous Integration & Delivery. By automating the way systems are configured, and how DIMS developed software is installed on them, not only are incremental changes possible with little effort, but multiple instances can be supported. Code that reaches the `master` branch is considered stable and release ready, at which point it can be pushed to test/evaluation and/or production systems. Development test systems would be fed by less stable branches (e.g., the `develop` branch.)

Documentation follows the same continuous integration and agile methodologies, using the Sphinx program, which processes ReStructured Text (reST) files (and is supported by the online documentation repository, ReadTheDocs.)

Use of Distributed Configuration Management

At the initiation of the DIMS project, the program Ansible was chosen for distributed system configuration and DIMS service deployment. Use of Ansible in DIMS is described in Section ansibleplaybooks:ansibleintro of ansibleplaybooks:ansibleplaybooks.

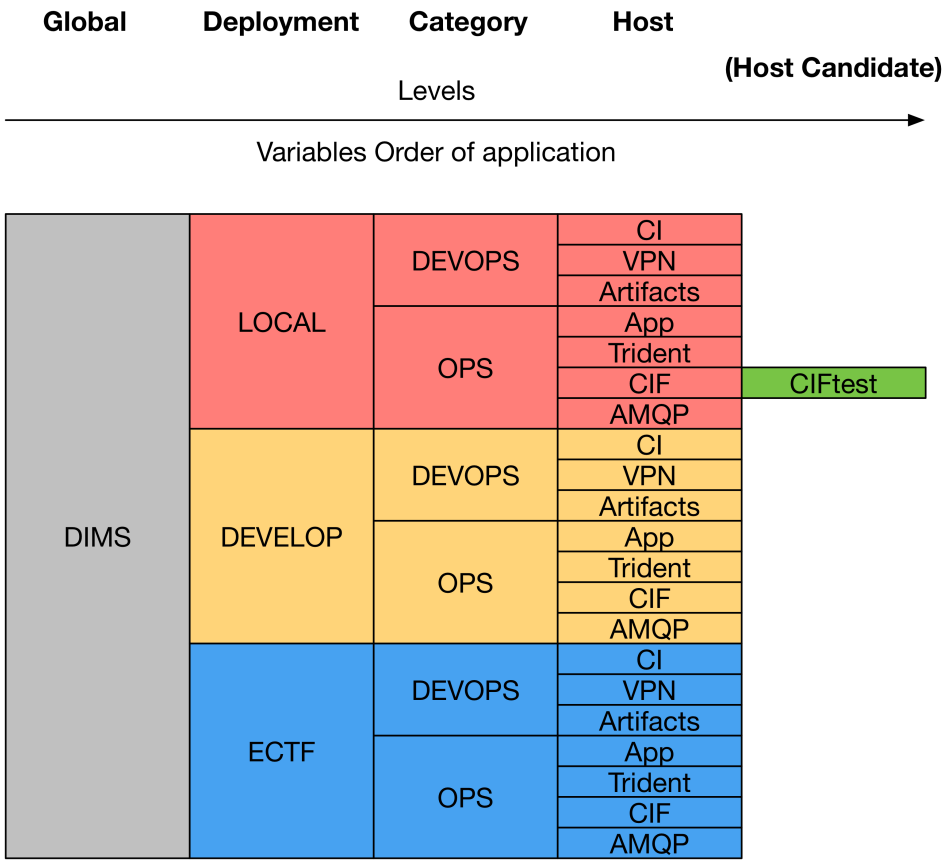


Fig. 3.10: Configuration Description Taxonomy

Figure *Configuration Description Taxonomy* illustrates the taxonomy of inheritence levels, following a left-to-right order of application of variables using global, `group_vars` and `host_vars` files (potentially augmented by playbook-level `vars` files for specific services.)

Attention: Setting variables in Ansible is quite complicated and should be studied and understood well by anyone attempting to construct playbooks or configure hosts and services. The ability to gain insight into how variables are set at runtime is crucial. The ansibleplaybooks:ansibleplaybooks documentation covers this topic.

Use of Containerization

During the Base year of the DIMS project, the focus was on taking as many open source tools as possible, and code developed by the DIMS team, and installing it on virtual machines using:

- Ubuntu (versions 10.04, 12.04, and 14.04), CentOS 5 and 6, and Mac OS X as host operating systems;

- Virtualbox and KVM as hypervisors;
- Packer for turning operating system installation ISOs into Box files for Virtualbox;
- Vagrant for provisioning virtual machines on developers' host operating systems of choice;
- Ansible for compiling code, configuring operating systems and services, installing pre-requisites libraries and tool dependencies, and other required DIMS tasks.

The team ran into a series of endlessly repeating problems that made progress painstakingly slow. These included:

- One person could get something running, only to hand it over to someone else to test (who could not run it).
- One team member could compile and install a program (because they had set up their system before hand with the requisite sofware), but another ran into missing dependencies and was blocked, not knowing what to do to get past the block.
- One team member could check in source code, only to find that another team member could not check it out because they had an out-of-date Git client.
- One team member could build a virtual machine with an open source package on it, but another did not know how to replicate the steps in the right order and could not get it to run.
- One team member would research a task, complete coding of Ansible playbooks to install the given software, but nobody else on the team could test it because they did not know the code existed or how to invoke it.
- One team member would code solutions to a problem that prevented widespread deployment of a given capability (such as component tests, status information collection, or event logging), but others on the team were not aware of the need to update their own development environments and things that formerly worked for them would "break".
- Frequently, only one team member was expert in a particular software package or operating system, but nobody else was. This made the person who knew how to do something a blocker in the critical path. If they were not available when someone else was trying to meet a deadline, the block would halt progress.
- Even when things worked right, and complete Vagrant virtual machines could be built and run with specific services running within them, IP addresses had to be configured by hand, and no DNS service existed that knew how to serve those IP addresses from domain names. This made it difficult for the team to know how to link services together, so things only worked when all software was installed in a single virtual machine (assuming that conflicting dependencies for libraries and operating system did not prevent all the software components from running on the same virtual machine.)

The result was what seemed like an endless chain of blockers that introduced friction throughout the entire process.

Concept for a new or modified system describes the operational concept for a new system, the DIMS framework model, which requires a mechanism that avoids the problems described above. The best available solution to these problems appears to be the use of *containers* (also known as Operating-system-level virtualization, or Microservices architecture).

Docker is seen as the leading technology in this area, garning a tremendous amount of support and energy. Docker is, "an open source project designed to easily create lightweight, portable, self-sufficient containers from any application." Their motto is "Build, ship, and run any application, anywhere." One of the main benefits of the use of containers is getting away from "dependency hell" of trying to fit a *least-common-denominator* of:

- *operating system* +
- *OS version* +
- *specific libraries* +
- *specific programming languages* +
- *specific dependant programs* +

- *specific service configuration settings*

Docker containers are not the perfect solution, by any means. There are certain security concerns, issues with linking containers together, keeping them up and running in the face of uncaught exceptions, etc. (Many of these same problems exist with use of bare-metal or virtual machines, so certain challenges remain regardless.) Figure *Run Services with Docker* (from https://coreos.com/using-coreos/) illustrates a 3-tiered web application in a clustered containter deployment.

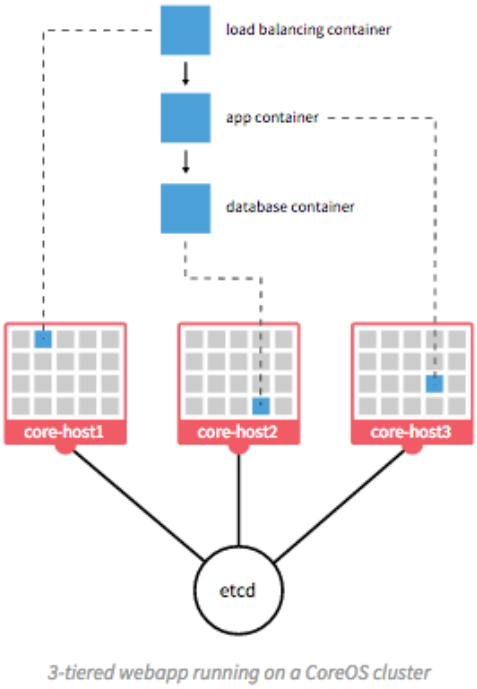


Fig. 3.11: Run Services with Docker

The suite of tools for orchestration, shared container components used to build higher-level images, distributed configuration and service discovery, persistent storage across clustered systems, domain name services, logging, and monitoring across a vast number of systems, all put Docker in a strong position in terms of open source software as opposed to virtual machines and the equivalent tools to manage large numbers of virtual machines. (The commercial tools supporting these tasks on virtual machines are out of the price range of SLTT government entities, let alone small- and medium-sized businesses and volunteer incident response groups.)

Note: For more information on all of these topics, see the Containerization, Virtualization, "Microservice Architectures" section of the PI's home page and the document dimsdockerfiles:usingdockerindims.

Use of Open Source components

Summary of High-Level System Architecture Delineation

At the beginning of this section in *Background on Existing Core Components* we saw DIMS from the perspective of data flows and core software components. A more detailed exposition of these components is found in DIMS Operational Concept Description v 2.9.0, Section Description of current system or situation.

In this section the focus is on delineating the components that are used to build the DIMS system from those that are functional in an operations context. Further, it will clarify the difference between the boxes on the left of Figure *Overview of DIMS System* (which have a subset of features that would be used by a non-operations investigative entity (e.g., US-CERT, the United States Secret Service, the Federal Trade Commission, or a Fusion Center) vs. the gray box in the bottom right of Figure *Overview of DIMS System* that includes the full set of realtime event data collection and network flow monitoring features that are more operational in nature.

A deployment of the core components of DIMS for a user such as the a law enforcement agency, a Fusion Center, etc, is depicted as *DIMS-OPS* in Figure *DIMS Operations*.

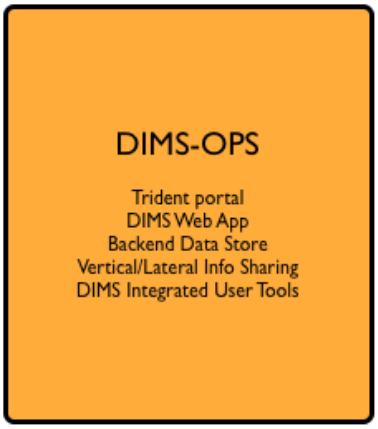


Fig. 3.12: DIMS Operations

Table 3.1: DIMS-OPS Components

Component	CSCI/Requirement
Trident portal and wiki	Backend Data Stores (BDS) CSCI, Design and implementation constraints
DIMS Web App	Dashboard Web Application (DWA) CSCI
LDAP Single-Signon	Data Integration and User Tools (DIUT) CSCI, [[networkAccessControls]] Network Access Controls
Redis, Hadoop (HDFS), Elasticsearch, etc.	Backend Data Stores (BDS) CSCI
OpenVPN	Data Integration and User Tools (DIUT) CSCI, [[networkAccessControls]] Network Access Controls
Tupelo	Data Integration and User Tools (DIUT) CSCI
Anonymization	Data Integration and User Tools (DIUT) CSCI
STIX input/output	Vertical/Lateral Information Sharing (VLIS) CSCI

Adding in the realtime event data collection elements, known as *DIMS-PISCES* is illustrated in Figure *DIMS Operations + PISCES*.[1]

[1] The term *PISCES* is the proposed replacement for *PRISEM* moving forward.

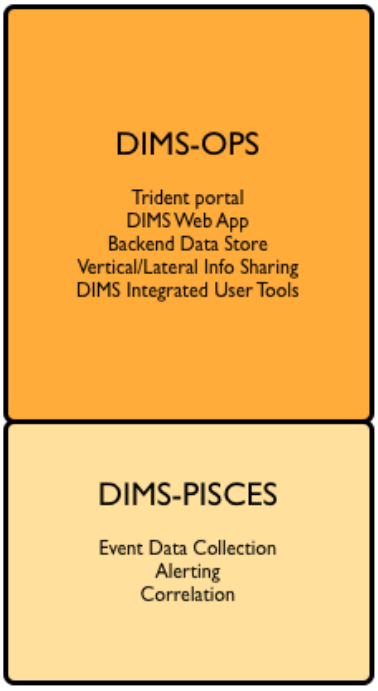


Fig. 3.13: DIMS Operations + PISCES

Table 3.2: DIMS-PISCES Components

Component	CSCI/Requirement
Distributed Security Event Data Collection	Backend Data Stores (BDS) CSCI
Alerting	Data Integration and User Tools (DIUT) CSCI, Dashboard Web Application (DWA) CSCI
Cross-organizational Correlation	Data Integration and User Tools (DIUT) CSCI, Dashboard Web Application (DWA) CSCI
Customized User Documentation	Adaptation requirements
Custom Configuration and Automated Deployment	Adaptation requirements, [[automatedProvisioning]] Automated Provisioning, [[continuousIntegration]] Continuous Integration & Delivery

Finally, the DIMS team (or anyone wishing to develop DIMS from the open source code base) requires all of the code development, configuration management, and continuous integration (or *DevOps*) features necessary for development. This is illustrated in Figure *DIMS Operations + PISCES + DevOps*.

Table 3.3: DIMS-DEVOPS Components

Component	CSCI/Requirement
Trident portal and wiki	Backend Data Stores (BDS) CSCI, Design and implementation constraints
Git source repository management	Design and implementation constraints
Jenkins Continuous Integration	Design and implementation constraints
Ansible configuration	Design and implementation constraints
Distributed configuration database	Backend Data Stores (BDS) CSCI, Design and implementation constraints
Docker repository	Backend Data Stores (BDS) CSCI, Design and implementation constraints
Jira ticketing	Design and implementation constraints

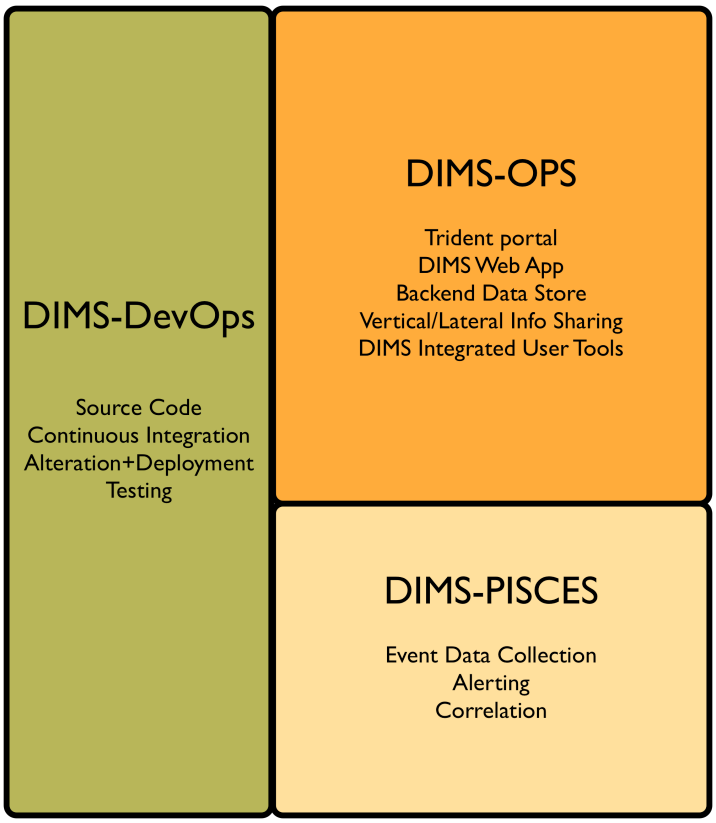


Fig. 3.14: DIMS Operations + PISCES + DevOps

For a pilot deployment of DIMS for the U.S. Secret Service, a full DIMS-OPS + DIMS-PISCES deployment will be instantiated for a select subset of the PRISEM participants in the Puget Sound to replicate a group of "victim" sites. Using live data, an incident will be investigated and "reported" to a test "U.S. Secret Service" DIMS-OPS system. This will validate the concept of reporting machine-parsable data to a central site using the Vertical and Lateral Information Sharing CSCI components (see Vertical/Lateral Information Sharing (VLIS) CSCI and DIMS Test Plan v 2.9.1).

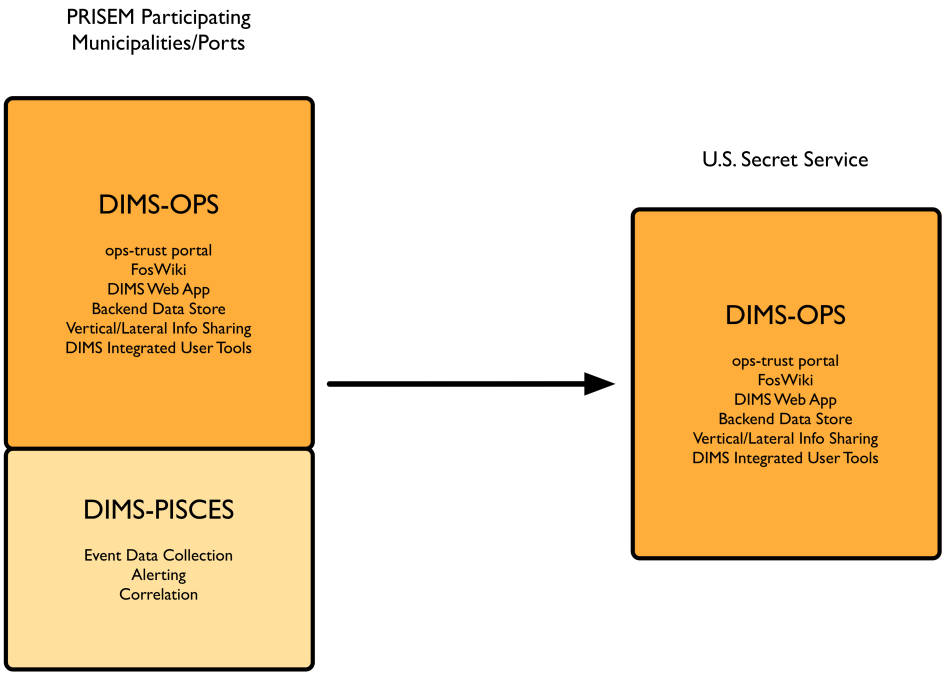


Fig. 3.15: U.S. Secret Service Pilot

CHAPTER 4

DIMS architectural design

Figure *DIMS Integrated System Architecture* illustrates the combined systems of the PRISEM project, the Trident portal (formerly the *Ops-Trust portal*), and the DIMS back end. As much as possible, the DIMS architecture was built to be overlaid on top of, or merged into, similar components from these existing systems. For example, it is not necessary to run three DNS servers for each project, when one can handle multiple systems and possibly even multiple domains. These can thus be collapsed into one server for DNS. The same is true for LDAP and OpenID authentication (the Trident portal and DIMS are both designed to use these services) and there is only need for one AMQP message bus server, one mail server, and one database for security data. All access will be centralized through the OpenVPN server, with certificates and encryption keys provided to the user via the Trident portal.

Note: This document was originally written prior to the Ops-Trust portal being renamed *Trident*. The name *Ops-Trust portal* may still exist throughout this, and related DIMS documents. An effort has been made to cross-reference the new portal name where possible.

System Software Architecture

The DIMS system conforms with the hardware/software separation used by the Trident and PRISEM systems, which pre-date the DIMS project. In both of these projects, some separation of services across physical and/or virtual machines was done for various reasons of security, performance, scalability, ease of administration, conformance with operating system version dependencies, etc.

SIEM event correlation server

The PRISEM system uses a Log Matrix "Threat Center" system, hosted on a high-end Dell server with multiple cores, large RAM capacity, an SSD drive to accelerate database activities, and 2TB RAID 1 array for disk fault tolerance. This system runs CentOS 6.4.

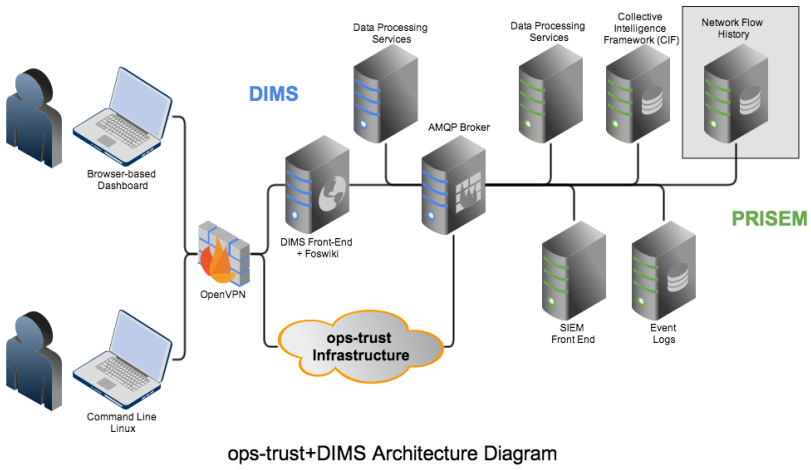


Fig. 4.1: DIMS Integrated System Architecture

SIEM log archive server

The PRISEM system uses a Log Matrix "Log Center" system, hosted on a high-end Dell server with multiple cores, and 9TB RAID 5 array disk fault tolerance. This system runs CentOS 5.10 (due to compatibility issues with the Vertica database).

Virtual machine management server

The PRISEM system uses a Dell PowerEdge R715 for virtual machine hosting.

AMQP broker

The PRISEM system uses a virtual machine running RabbitMQ for AMQP broker services.

Collective Intelligence Framework (CIF) server

The PRISEM system is using a CIF v0.1 database on physical hardware (Dell PowerEdge 1950). This system will be replaced with a virtual machine running CIF v1.0 (or newer).

ID management and authentication server

The Ops-Trust and DIMS projects are using OpenID and LemonLDAP (though in slightly different ways). The intention is to combine these into a single pair of OpenID/LDAP servers.

Domain name server

The Ops-Trust system runs its own DNS server for all system components in a single-purpose VM. The PRISEM project is currently using static host tables and DNSMasq in slightly different ways (depending on whether access is

from the open internet, or through the OpenVPN tunnel). It is anticipated that a split-DNS configuration, using the same server as the Ops-Trust infrastructure, will be used in the long run to get consistent DNS response regardless of access method used.

Virtual private network tunnel server(s)

The PRISEM project has an OpenVPN server for remote access to the backdoor VLAN. This allows access to the AMQP broker, and direct access to the MySQL databases used by Log Matrix. (The vendor maintains their own Cisco managed VPN/firewall for access from their internal network).

DIMS components are separated (as appropriate) for similar reasons, and integrated as much as possible by combining similar services in order to minimize the total number of physical and/or virtual machines in use. For example, if there are three domain name servers, they can be combined into one server that handles multiple domains.

The following Figures help illustrate the concepts of system implementation of the service components in relationship to physical ("bare-metal") hardware.

Of course the simplest design is to take a hardware system, install a single operating system on it, and install every piece of software into that single server. For the reasons listed above, this is not a viable or practical solution, since the component pieces were never designed to work this way. The level of effort required to debug, patch, document, and attempt to get the original authors to accept the code changes into their code base (to avoid adding a maintence cost for maintaining your locally patched fork as new versions of the original software are released) is not sustainable.

Figure *Pure Virtual Machine Architecture* shows the next simplest design, which is to host multiple virtual machines on a single server. The *Host* is shown at the bottom, comprised of a highly-provisioned server, a base operating system and a virtual machine hypervisor. Each virtual machine *Guest* is then created and installed with its own combination of base operating system, libraries and binaries, and application software. The result is a single physical computer with a total of six servers (4 Ubuntu Linux, 1 Red Hat Enterprise Linux, and 1 Debian Linux) that must be configured, patched, and maintained separately. There is also a higher overhead for processing and memory, due to the hypervisor hardware virtualization layer.

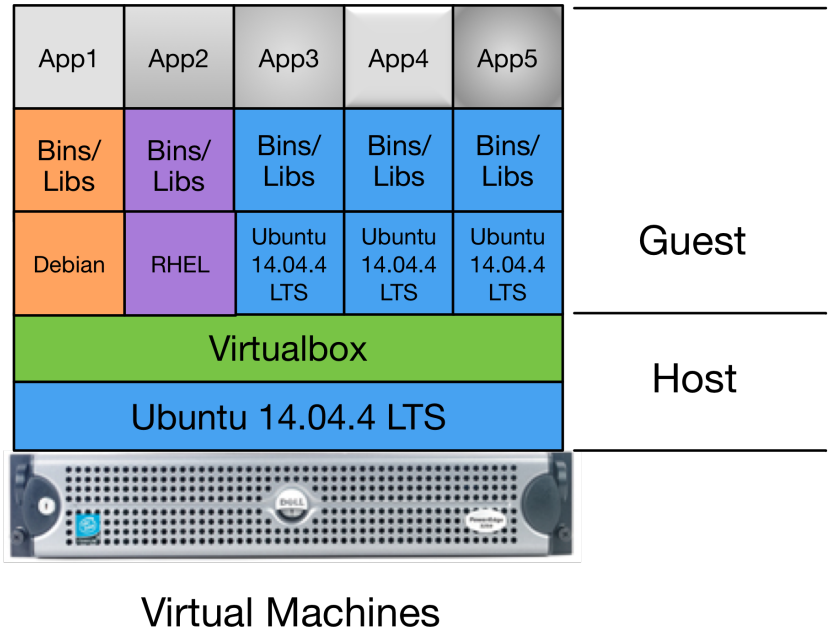


Fig. 4.2: Pure Virtual Machine Architecture

Figure *Pure Container Architecture* shows an alternative to virtual machines, which is the use of pure *containerization* of all services. In this architectural model, the Docker Engine replaces the hypervisor as part of the Host layer. On top of this are *Containers* formed from images that combine the foundational operating system bits, libraries, binaries, and application software as in Figure *Pure Virtual Machine Architecture*, except there is no virtualization of hardware taking place. Instead, container images hold the file system contents that are then executed in isolated process spaces that function similarly to virtual machines (though simpler, as they only provide a service oriented function, rather than a fully-functional operating system into which you log in to like a "normal" virtual machine or bare-metal operating system.) One of the principle advantages to this architectural model is the separation of content in each container, allowing just the specific base operating system, binaries and libraries, and application code for each open source security tool to be set up exactly as the producer supports with no risk of that breaking other tools that are part of the larger system.

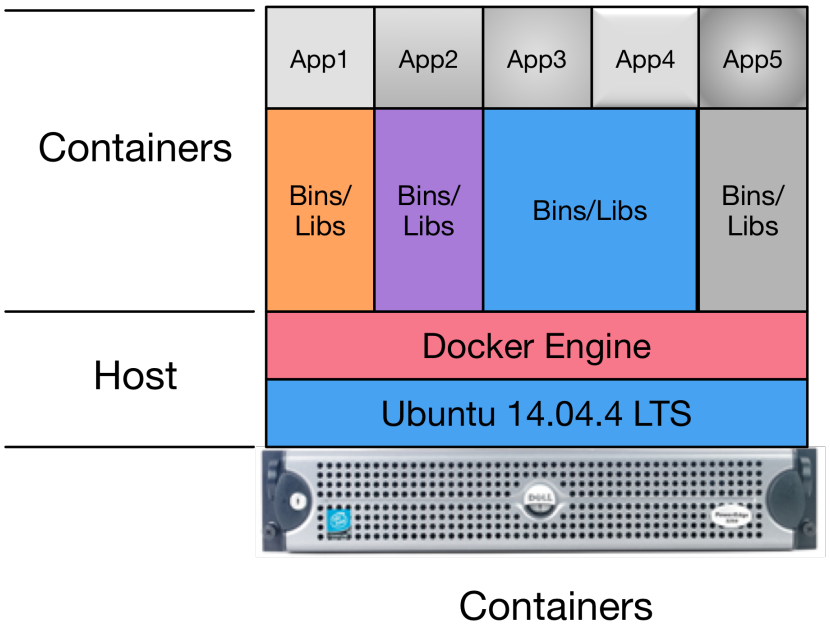


Fig. 4.3: Pure Container Architecture

Of course is it much more complicated than this in real life. Requirements for scalability and redundancy drive towards use of multiple bare-metal servers. This leads to two more architectural models to add to the mix.

The first is to clusterize the containerized model we just saw. Figure *Clusterized Container Architecture* depicts a two-node CoreOS cluster. A program `etcd` is used as a distributed key/value store that facilitates managing the distribution and management of containers across the two server nodes. *App2* and *App3*, in this case, have multiple instances running (2x App2 and 3x App3, in this case), both split across the two cluster members. This allows one of the two cluster servers to be taken off-line without disrupting the services provided by App2 and App3.

The final architectual model is combination of the earlier models. Figure *Hybrid VM+Container Architecture* depicts a *Hybrid* combination of bare metal, virtual machines, and containers within virtual machines. (Because containers are so light-weight, you can run containers in both the Host and Guests, containers within containers, or combinations nested within each other!)

The hybrid model can be accomplished by adding virtualized CoreOS nodes in the form of VM Guests along side other VM Guests in one server (as shown in Figure *Hybrid VM+Container Architecture*), or splitting pure virtual machines and pure-containerization across multiple servers (as shown in Figure *Multi-server Hybrid Architecture*).

A primary advantage of this architectual model is the ability to use the separation of virtual machine Guests to leverage

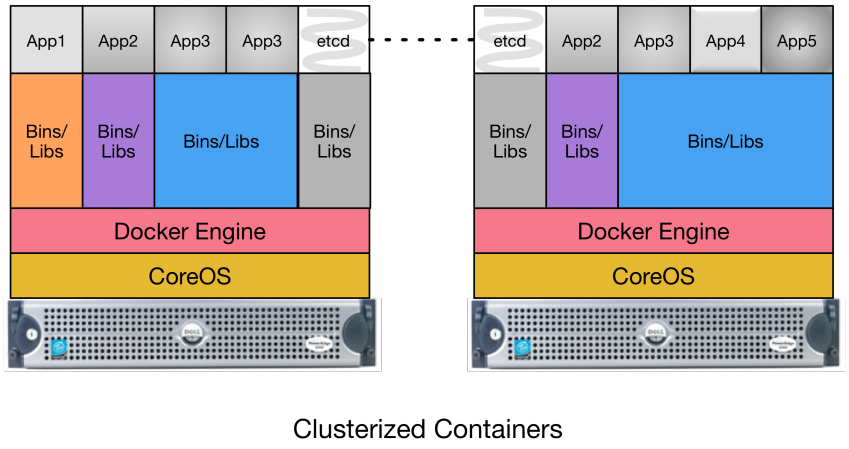


Fig. 4.4: Clusterized Container Architecture

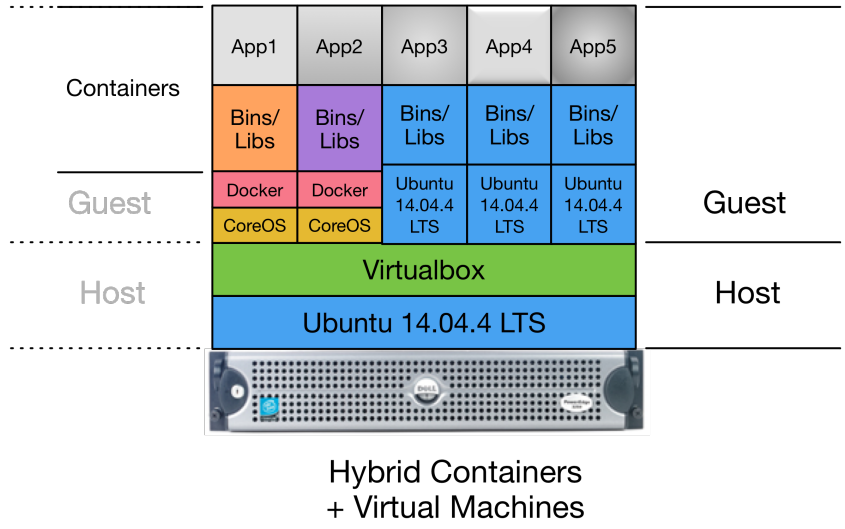


Fig. 4.5: Hybrid VM+Container Architecture

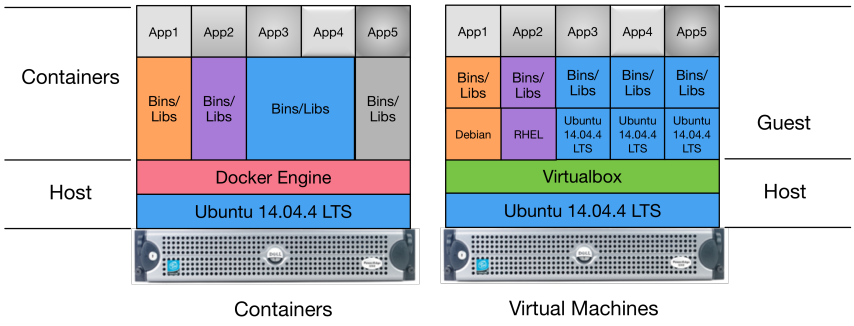


Fig. 4.6: Multi-server Hybrid Architecture

operating systems like CoreOS (that are designed for clusterized containerization support) along side multiple different base operating systems in other virtual machine Guests, within a single hardware server. This allows movement of services as necessary to address performance issues that may be discovered over time as the system scales up.

Note: The architecture currently being used for DIMS development uses the Hybrid model, with a three-node CoreOS cluster on bare-metal servers, with a fourth server supporting virtual machine Guests.

Work is underway to replicate the entire system using a single-server Hybrid deployment as shown in Figure *Hybrid VM+Container Architecture* as the prototype for the U.S. Secret Service ECTF deployment.

The ECTF deployment is being planned to be done with two servers with a subset of DIMS components (primarily focusing on the Trident portal.) One server will be used to provide a stable "production" service platform, while the second server can be used for staging new releases, testing, supporting migration to other data center facilities, or as a fallback in case the first system is damaged.

Internal Communications Architecture

The DIMS system was designed to overlay on top of the legacy PRISEM system and other open source security tools. PRISEM had interfaces to some of its services that integrated an instance of the Collective Intelligence Framework (CIF) database for IP-based reputation watchlists and historic attacker context, an archive of historic event logs, and remotely stored network flow data in SiLK format. The logical architecture that integrated these systems is a combination of message bus (using AMQP), SSH tunneled file and/or command line access, or HTTPS web interfaces and RESTful API.

Figure *AMQP Messaging Bus Architecture* shows the general flow of commands and logged events from clients and services used in the PRISEM system for inter-process communication between system components. In this example, there are three general RPC services named *A*, *B*, and *C*. Calls from remote clients *A* (color blue) and *B* (color black) are processed by one of n instances of multiprocessing service daemons on the same hardware as the AMQP broker (by multiple processes or virtual machines). Client *C* in this diagram (color green) is also a remote client, as is the RPC service *C*. (The AMQP broker and RPC mechanism allows these programs to run anywhere we want.) Also depicted in this diagram is an event feedback loop (color red). All clients and services log significant events such as process startup, process end, time taken to process RPC calls, or even more fine-grained debugging output to assist developers. These events logs are published to a fanout exchange, which distributes the events to any subscribers who wish to consume them.

Figure *Remote access via OpenVPN to VLAN1* depicts a high-level view of remote access from developer laptops (on the right) or servers at a remote site (on the left) using an OpenVPN tunnel that is routed via Network Address Translation to a non-public VLAN. This simplistic diagram does not show specific routable IP addresses of the remote systems, though it does show the tunnel IP address assigned by OpenVPN in relation to the OpenVPN server, and the difference between the network address ranges used by hosts on VLAN1 vs. the OpenVPN tunnel.

Note: In reality, there are multiple non-private network address ranges and VLANs in use by Virtual Machine hypervisors, Docker containers, and physical switch VLANs. This is described in the "DIMS As-Built" document and we are in the process of simplifying the highly-complicated networking implementation that resulted from building on top of the legacy PRISEM platform that goes back to the project's initiation in 2008.

Concept of execution

The problem of event collection, correlation, and alerting, is quite common. Nearly every anti-virus vendor, managed security service provider, major internet platform provider, or multi-national enterprise, shares similar problems with

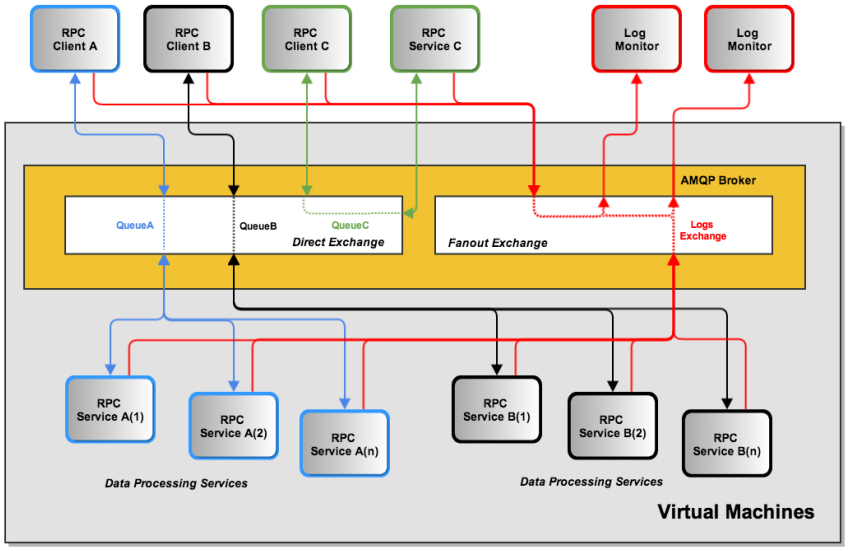


Fig. 4.7: AMQP Messaging Bus Architecture

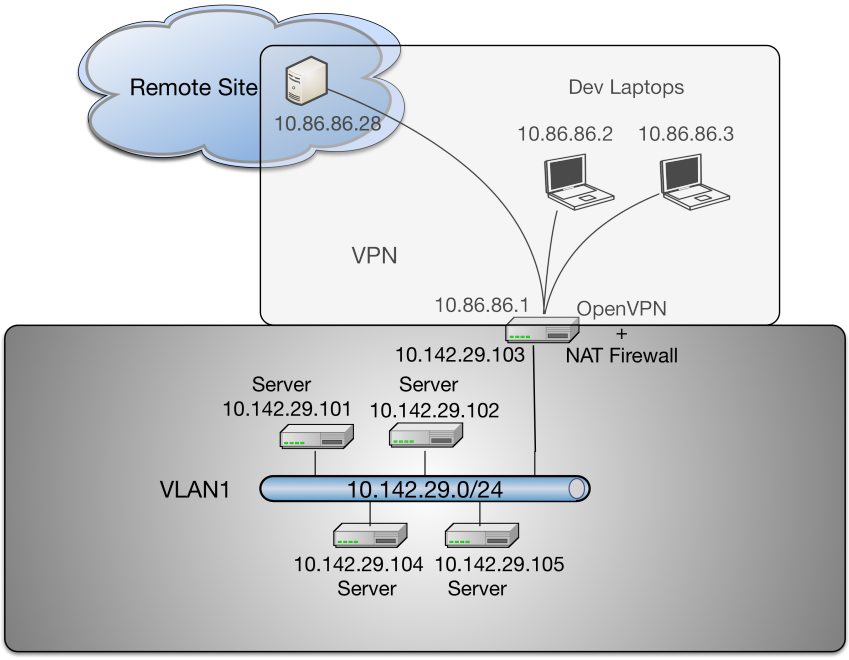


Fig. 4.8: Remote access via OpenVPN to VLAN1

processing event data. They are nearly all looking at the same type of architecture to handle the high data volumes and flow rates associated with large, high-bandwidth networks.

A common combination of open source tools used to process and index large volumes of event logs is *Elasticsearch*, *Logstash*, and *Kibana*, known as the "ELK stack" for short. The developers of the ELK stack refer to it[2] as "an end-to-end stack that delivers actionable insights in real-time from almost any type of structured and unstructured data source." Elasticsearch provides flexible storage of data and flexible search of data. Logstash is used to parse the data, and then it sends it to Elasticsearch. Kibana then takes the parsed data from Elasticsearch and presents it through a browser in an easy-to-view way. Kibana's dashboards are customizable in a variety of ways to we can better dissect and view the data.

- Elasticsearch
- Architecture behind our new Search and Explore experience (where "our" is Soundcloud... see "final box-diagram")
- How HipChat Stores And Indexes Billions Of Messages Using ElasticSearch And Redis
- Using elasticsearch and logstash to serve billions of searchable events for customers
- Example configuration of Elasticsearch for AOL's Moloch network flow monitoring tool
- How to use Elasticsearch with Python
- Security Analysts Discuss SIEM'S – Elasticsearch/Logstash/Kibana vs ARCSight, Splunk, and more
- Scaling an ELK stack at bol.com
- Logstash
- What is Logstash?
- Github logstash/cookbook
- Kibana
- Creating an Advanced Kibana Dashboard Using a Script
- Templates and Scripts
- Command Line Load Dashboard

The ELK stack has been used to process hundreds of millions to billions of events per day. Mozilla uses it as part of the Mozilla Defense Platform, or MozDef. (See Figure *MozDef data flows* for the data flow diagram for the Mozilla Defense Platform, or MozDef. See also Jeff Bryner's Bsides PDX 2014 presenation on MozDef and Anthony Verez' presenation MozDef: You've collected your security logs, now what? and accompanying slides.) The company Mailgun has described how they are Using elasticsearch and logstash to serve billions of searchable events for customers. (For an order of magnitude comparison, the PRISEM system currently collects between 30-60 million events per day, not the billions described in this reference.)

Figure *Logstash and Metrics* (source[1]) shows how the event log collection process works in terms of data flows between sources and ELK stack components. The DIMS system is designed to sit on top of such an event collection infrastructure.

Section *Use of Containerization* discusses Docker and its role in implementing a *micro-service architecture*. ELK stack components have been demonstrated being implemented in containers. (E.g., see Automating Docker Logging: ElasticSearch, Logstash, Kibana, and Logspout, by Nathan LeClaire and Scalable Docker Monitoring with Fluentd, Elasticsearch and Kibana 4, by manu, Elasticsearch, Weave and Docker, by errordeveloper, and the GitHub repository of iantruslove/docker-elasticsearch with a Docker image for ElasticSearch using Maestro orchestration.)

[2] http://www.elasticsearch.org/overview
[1] http://www.semicomplete.com/presentations/logstash-hmmm

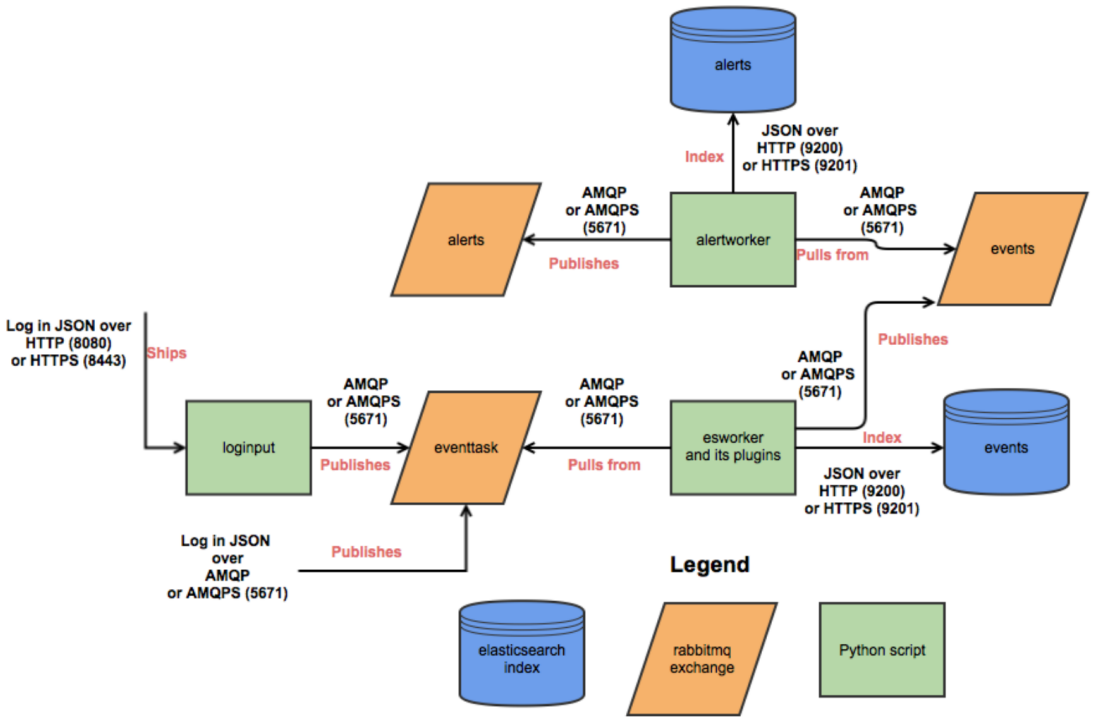


Fig. 4.9: MozDef data flows

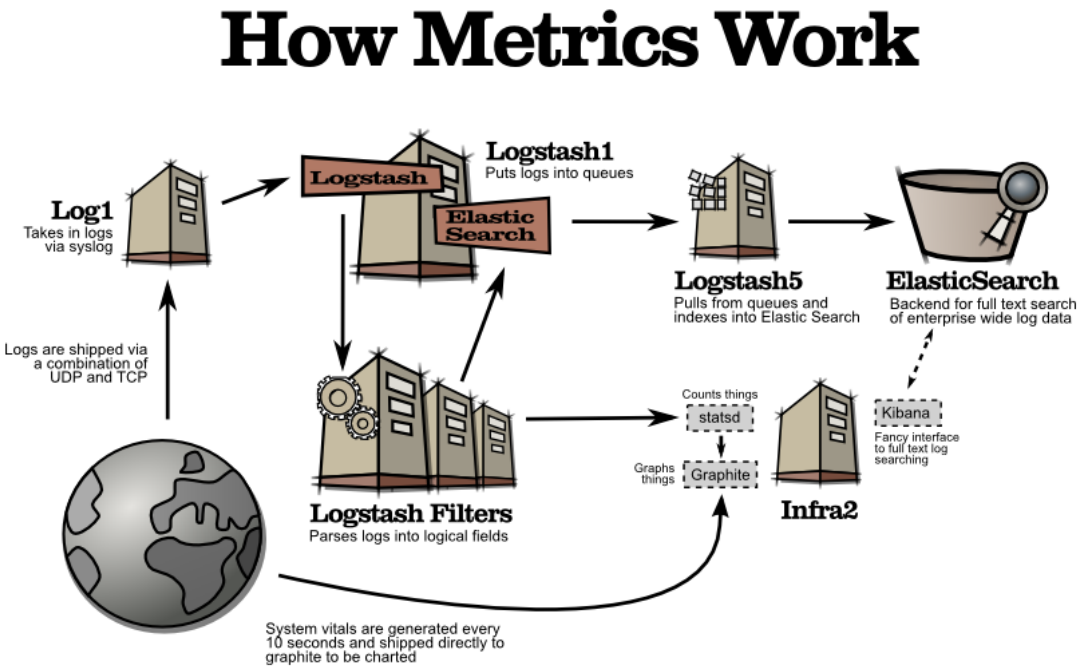


Fig. 4.10: Logstash and Metrics

Real-time monitoring of Hadoop clusters describes deploying the ELK stack alongside Hadoop cluster nodes to provide a realtime monitoring capability. (See also Apache Hadoop 2.6.0 on Docker, by Janos Matyas, for containerizing the Hadoop cluster nodes.)

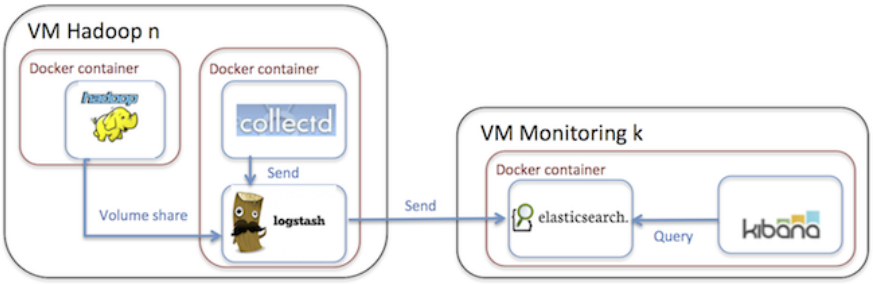


Fig. 4.11: High-level architecture for monitoring Hadoop with the ELK stack

Interface design

File and Database Design

System	Data Type	Volume	Time covered
ThreatCenter	Real-time events	~40M records/30GB	48 hours
LogCenter	Historical logs	186,772 files/2.1TB	1 year
SiLK records	Network flows	258,000 files/803GB	2+ years
CIF	Threat data feeds	4.9M archived records/22GB	(NA)

Fig. 4.12: PRISEM Data Volumes

Figure *PRISEM Data Volumes* lists the database and non-database data sources used by the PRISEM system, along with the approximate timespan over which those records are kept.

Database Management System Files

There is an approximate average of 20M events per day collected by the ThreatCenter database server (zion.prisem.washington.edu), which is configured with a 48-hour data retention window. These records are kept in a database optimized for continuous correlation. The normalized records (which include the original raw event log) are stored in over 167,000 discrete read-optimized Vertica database files on the LogCenter server (money.prisem.washington.edu). The Collective Intelligence Framework database (v0.1) keeps its data in a Postgress database. This database is used to pull feeds from remote sites, and to generate feeds for use by the Botnets system's watchlist detectors. At regular periods during the day, the CIF database has some tables copied into a read-optimized MySQL database known as Sphinx for accelerated discrete queries. (It is the Sphinx database that is used by the cifbulk RPC service).

Non-Database Management System Files

Network flow records are stored locally at the City of Seattle (pink.seattle.gov) in SiLK format. The disk capacity of 1TB is capable of holding just over 2 years of flow data in over 258,000 discrete SiLK data files. (SiLK is a highly-optimized fixed length binary format that is quite efficient for post-processing without needing a database management system.)

Human-Machine Interface

The raw inputs to PRISEM fall into three primary buckets: event logs from security devices in text form, which are normalized as they are processed by the SIEM; Network flow records that are received as NetFlow V5 records processed in real time and discarded, but a copy is converted to SiLK format and saved for historic query capability; reputation data pulled from various feeds and stored in a Collective Intelligence Framework (CIF) database. Various ad-hoc formats of "indicators of compromise" or "observables" are received from outside parties, which are primarily processed by hand (this includes indicators received from federal government sources, for example Joint Indicator Bulletins (JIBs) from the Federal Bureau of Investigation).

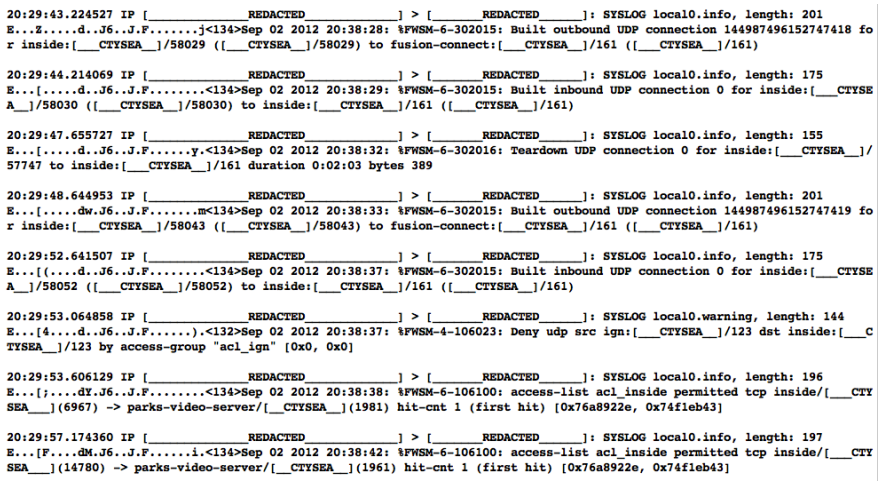


Fig. 4.13: Cisco FWSM Event Log (Redacted)

Examples of standard security device logs can be seen in Figure *Cisco FWSM Event Log (Redacted)* (Cisco Firewall Security Manager, or FWSM), Figure *Netscreen Event Log (Redacted)* (Netscreen Firewall), Figure *Tipping Point Logs (Redacted)* (Tipping Point Intrusion Prevention System, or IPS), and Figure *WebSense Log Sample (Redacted)* (Websense web filter). These examples are redacted, but show representative content that is used for correlation (e.g., source and destination IP addresses, ports, protocols, etc.)

Figure *Botnets System Event Log (Redacted)* illustrates what events logged by the Botnets system detectors look like. All of these examples are for "watchlist" detectors that simply trigger when they see a connection to/from a host on the watchlist. Each detector has its own ID (e.g, "CIFList" in the first entry), followed by the ranking score for that detector ("@8" in this case for the CIFList detector). This is used in the calculation of score for ranking significance of events in the SIEM. Also shown are the IP addresses of the internal hosts involved in the alerted activity, as well as the IP addresses of the systems on the watchlists.

Figure *Example Historic Event Log Data (Redacted)* shows three records returned from a search of historic event logs from the Log Matrix SEIM log archive. These records have been anonymized to conceal the specific IP addresses and domain names of the sources (Seattle Children's Hospital and the Port of Tacoma, in this case). Notice that the schema used by this vendor includes both destination IP address and destination port, but only includes source IP address (not

Fig. 4.14: Netscreen Event Log (Redacted)

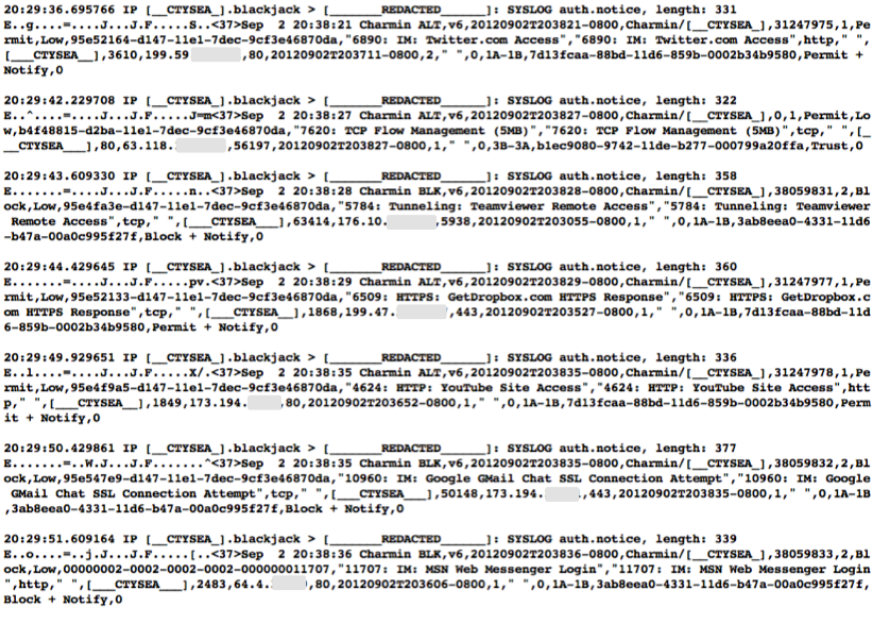


Fig. 4.15: Tipping Point Logs (Redacted)

Fig. 4.16: WebSense Log Sample (Redacted)

Fig. 4.17: Botnets System Event Log (Redacted)

```
[
    {
        "CUSTOM3": "Firewall",
        "description": "date=\"Feb  4 08:38:33 2014 UTC\",fac=f_generic_proxy,area=a_proxy,type=t_attack,pri=p_major,pid=2
544,ruid=0,euid=0,pgid=2544,logid=0,cmd=tcpgsp,domain=Genx,edomain=Genx,hostname=[________SEA-CHILD_______],category=pol
icy_violation,event=ACL deny,attackip=87.139.      ,attackburb=external,srcip=87.139.      ,srcport=57833,srcburb=extern
al,dstip=[__SEA-CHILD_],dstport=25,dstburb=internal,protocol=6,service_name=tcp25,user_name=(null),auth_method=(null),rule
_name=\"Deny All\",cache_hit=1,reason=\"T\",
        "device_vendor": "SecureComputing Firewall",
        "dstip": "[__SEA-CHILD_]",
        "dstport": 25,
        "event_name": "attack",
        "report_device": "ppxpfw01b",
        "srcip": "87.139.      ",
        "timestamp": 1391500829,
        "usr": null
    },
    {
        "CUSTOM3": "Firewall",
        "description": "date=\"Feb  4 08:38:46 2014 UTC\",fac=f_generic_proxy,area=a_proxy,type=t_attack,pri=p_major,pid=2
541,ruid=0,euid=0,pgid=2541,logid=0,cmd=tcpgsp,domain=Genx,edomain=Genx,hostname=[________SEA-CHILD_______],category=pol
icy_violation,event=ACL deny,attackip=210.186.      ,attackburb=external,srcip=210.186.      ,srcport=3013,srcburb=exter
nal,dstip=[__SEA-CHILD_],dstport=25,dstburb=internal,protocol=6,service_name=tcp25,user_name=(null),auth_method=(null),rul
e_name=\"Deny All\",cache_hit=1,reason=\"\",
        "device_vendor": "SecureComputing Firewall",
        "dstip": "[__SEA-CHILD_]",
        "dstport": 25,
        "event_name": "attack",
        "report_device": "ppxpfw01b",
        "srcip": "210.186.      ",
        "timestamp": 1391500843,
        "usr": null
    },
    {
        "CUSTOM3": "Firewall",
        "description": "2014/02/04 12:00:41,63.115.40.56,[__PORTTAC__],0.0.0.0,0.0.0.0,Untrust-To-DMZ189-SMTP,,,smtp,vsys1
,L3-Untrust,DMZ-189,ethernet1/1,ethernet1/3,POT SysLog Servers,2014/02/04 12:00:41,119244,1,56786,25,0,0,0x0,tcp,allow,146
9,550,919,17,2014/02/04 12:00:11,0,any,0,489843145,0x0,United States,United States,0,8,9",
        "device_vendor": "PAN",
        "dstip": "[__PORTTAC__]",
        "dstport": 25,
        "event_name": "allow",
        "report_device": "192.168.253.228",
        "srcip": "63.115.    ",
        "timestamp": 1391500847,
        "usr": null
    },
```


Fig. 4.18: Example Historic Event Log Data (Redacted)

source port) making certain queries of the database impossible. For example, attempting to find records related to malware that uses fixed source port for flooding could not be directly queried, requiring extraction of the "description" field (i.e., the original raw event) and parsing to identify related records. A solution to this would be to extract all of the data from the database and store it in a more flexible database.

Indirectly related to the previous data sources is meta-data that allows classification, filtering, and anonymization, based on organizational units for networks and sites. Table *Participant identification mapping* illustrates how top level domains and/or CIDR blocks for a subset of PRISEM participants are mapped to their Site ID strings and chosen anonymization strings (i.e., the label that participant would like to use to mask their internal IP addresses and host names in reports that are shared outside the trust group.) Their use in identification of "Friend or Foe" is described in the Concept of Operations document. (Such a cross-organizational correlation result using the full map as suggested in Table *Participant identification mapping* can be seen in Figure :ref'crosscorriff'.)

Table 4.1: Participant identification mapping

CIDR or Domain	Site ID	Participant
156.74.0.0/16	CTYSEA	CTYSEA
.seattle.gov	CTYSEA	CTYSEA
.seattle.wa.gov	CTYSEA	CTYSEA
.seattle.wa.us	CTYSEA	CTYSEA
192.103.189.0/24	PORTTAC	PORTTAC
66.113.101.0/24	PORTTAC	PORTTAC
.portoftacoma.com	PORTTAC	PORTTAC
174.127.160.0/24	COB	BELLWA
12.17.152.0/23	COB	BELLWA
.bellevue.gov	COB	BELLWA
.ci.bellevue.wa.us	COB	BELLWA

```
{"program": "crosscor",
 "date": "Mon Mar 10 18:27:51 PDT 2014",
 "iff": "friend",
 "matching": [
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"PORTTAC"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"KITSAP"},
   {"ip4":"[redacted]", "site":"KITSAP"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"KITSAP"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"PORTOLY"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"BELLWA"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"KITSAP"},
   {"ip4":"[redacted]", "site":"SEA-CHILD"},
   {"ip4":"[redacted]", "site":"CTYSEA"},
   {"ip4":"[redacted]", "site":"CTYSEA"},
   {"ip4":"[redacted]", "site":"BELLWA"},
   {"ip4":"[redacted]", "site":"BELLWA"},
   { ... }
 ],
 "matching_stats": [
   {"site":"PORTOLY","count":"58","percent":"1.38"},
   {"site":"BELLWA","count":"41","percent":"0.98"},
   {"site":"KITSAP","count":"1119","percent":"26.61"},
   {"site":"PORTTAC","count":"7","percent":"0.17"},
   {"site":"KING","count":"5","percent":"0.12"},
   {"site":"SEA-CHILD","count":"2971","percent":"70.65"},
   {"site":"CTYSEA","count":"4","percent":"0.10"},
   {"site":"ALLSITES","count":"4205","percent":"100"}
 ]
}
```


Fig. 4.19: Cross-organizational Correlation of Query Results (Redacted)

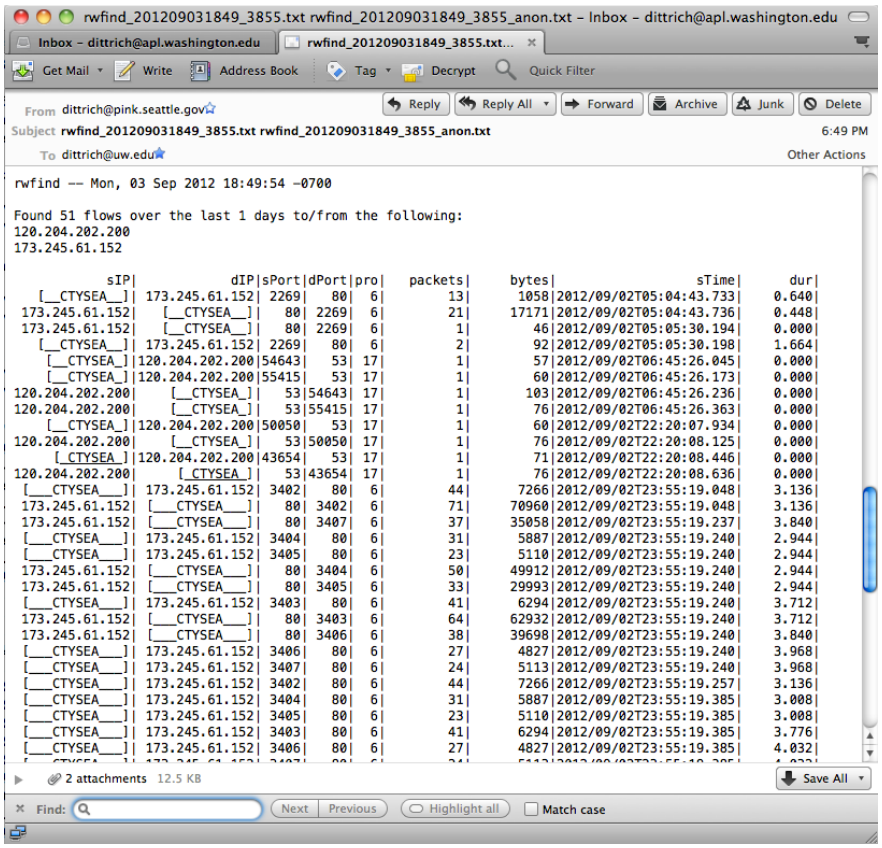


Fig. 4.20: Example Network Flow Report (Anonymized Targets)

CHAPTER 5

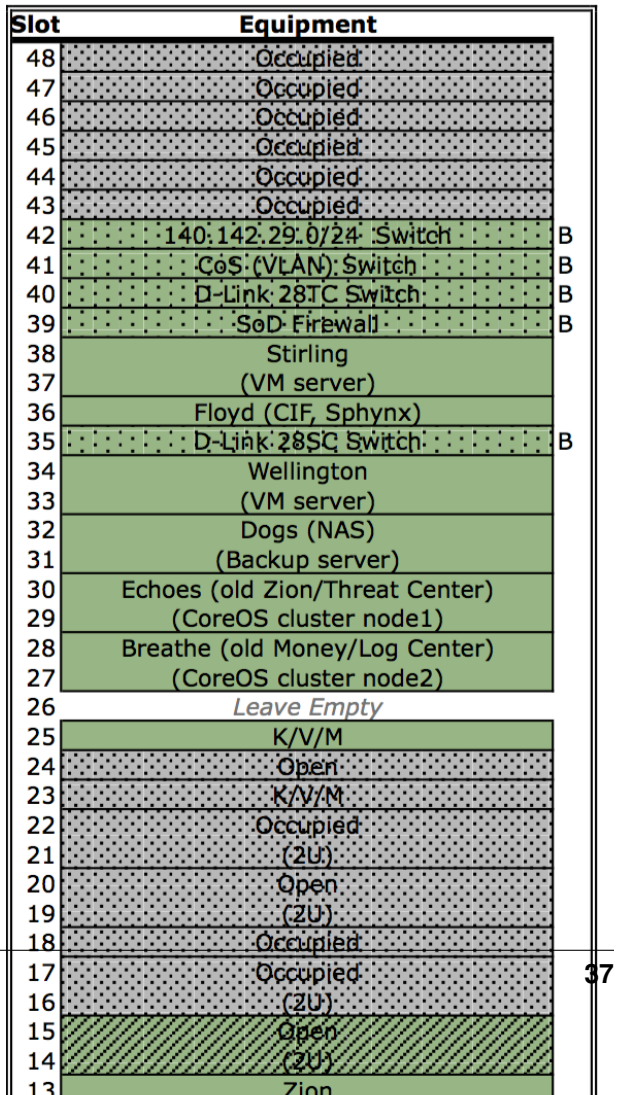
DIMS detailed design

Hardware Detailed Design

Figure PRISEMHardwareLayoutDiagram shows the physical hardware configuration for PRISEM system components in the server rack located in the UW Tower IT data center. Green boxes are those that were used for the PRISEM Project and for DIMS system development, while white and gray boxes are either unused or occupied by other resources. Some of the initial physical hardware that became unstable or obsolete was replaced was replaced by virtual machines.

The principal PRISEM hardware consisted of Dell PowerEdge servers. One PowerEdge 1950 server (`floyd`) was used for a CIF database server, and two Dell R720 servers (`zion` and `money`) servers were used for the Log Matrix Threat Center and Log Center servers. Both `zion` and `money` are replacements for the original Dell R710 servers purchased at the start of the project in 2008. Virtual machines are run on a Dell PowerEdge R715 server, with 128GB RAM, 2x12-Core 1.8GHz AMD Opteron processors, and 12 – 1TB drives in a RAID 5 array.

Physical networking is provided by multple managed switches, some configured to support virtual LAN (VLAN) isolation. One is a D-Link xStack Managed 24-Port Gigabit L2+ 1/10-GigE switch, another a D-Link DXS-3227 1-GigE managed switch. One VLAN provides an isolated network for inter-system communication behind a vendor-supported stateful firewall and OpenVPN server for remote access. Another VLAN



provides internet-routable connections in front of the firewall. At present, only IPv4 is supported for network connectivity.

Note: All of this is better detailed in the internal document dimsasbuilt:dimsasbuilt.

Software Detailed Design

The DIMS platform is made up of several open source sub-systems.

- A Dashboard web application (written using AngularJS) for workflow related operations. It provides a graphical user interface for control, with ReST style HTTP and Unix socket interfaces to backend services.
- A web application server (written using Node.js) that in Javascript) with the following interfaces:
  - HTTP - communicates with client
  - AMQP - communicates with AMQP server
  - Socket - communicates with client
  - Redis - communicates with redis database
  - Postgres - communicates with PostgreSQL
- An OpenID authentication and LDAP directory service that is used by DIMS components to provide a single-signon login mechanism.
- A RabbitMQ (AMQP) message bus for supporting remote procedure call services, and message brokering for things like chat and event logging.
- A Collective Intelligence Framework database server.

All of these open source components are installed and configured using Ansible from ad-hoc control hosts (e.g., developer laptops), and via a Jenkins continuous integration server by manual, or event-triggered, jobs.

Internal Communications Detailed Design

Figure *PRISEM AMQP Data Flows* shows a more detailed perspective on the central AMQP bus than that in Section *AMQP Messaging Bus Architecture*. Red boxes depict the command line clients, client applications, and "service" daemons that front-end accces to data stores (the gray boxes with solid Blue lines on top and bottom) and other command line programs (the Orange boxes). The dashed Green lines are TCP connections to the AMQP service port on the central RabbitMQ server on the host in the bottom left of the Figure. Because each of the Red boxes connects to the AMQP bus, it can *publish* or *subscribe* to data flows on specified named channels or exchanges. Programs wishing to use *services* publish their request in the form of special JSON command object, and they get back a special JSON response object with the results. (The details are described in the prisem:prisemdataquery Section of the prisem:prisemutilities document.

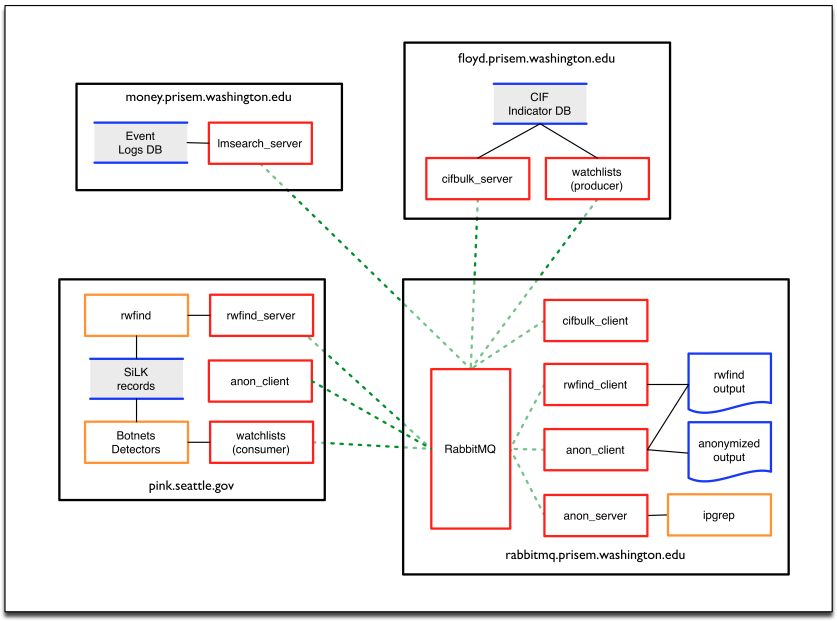


Fig. 5.2: PRISEM AMQP Data Flows

There are several services available within the PRISEM architecture as Remote Procedure Call (RPC) services, with some data distribution and feedback mechanisms in the form of publish/subscribe fanout services. These are:

- RPC service `rwfind` – This service provides search capability to stored network flow records kept in SiLK tools format. It returns the results in text report format for human consumption, or in structured JSON format for simplified processing by programs.
- RPC service `anon` – This service provides IP address and DNS name identification/anonymization/extraction, statistics, match/non-match identification, and other functions, using the `ipgrep` script. This service is called as part of the `crosscor` service in order to *identify friend or foe*.
- RPC service `cifbulk` – This service front-ends the Sphinx database accelerator, which provides a read-only snapshot of the CIF database for a 10:1 speed increase for queries. It takes as input a list of items to search for, and iterates over the list of items it is passed concatenating the results (which are JSON by design) into a JSON array.
- RPC service `crosscor` – This service performs cross-organizational correlation on search results obtained from the `rwfind`, `lmsearch`, and `cifbulk` services.
- Watchlist generation – Currently, a scheduled script produces watchlist files from CIF feeds and distributes them to systems that use the watchlists via `rsync` over SSH tunnels. These will be replaced, eventually, with publish/subscribe services via AMQP.
- Daily reports from the Botnets system – Currently, a scheduled script generates daily reports that summarize the detected activity by the Botnets system. This text report will be enriched with context provided by the `cifbulk` service, the `crosscor` service, and the *identify friend or foe* mechanism. This will be a model for a suite of DIMS scheduled reports.

Figure *DIMS and Trident Component Interfaces* depicts the communication flows between components within the DIMS code base, and those within the Trident (ops-trust portal re-write) code base at a logical level. Both DIMS and Trident have architecturally split their back end data stores from the front end user interfaces (each having a command line interface and a web application graphical user interface.)

DIMS components that need to communicate to the Trident backend user database can either use the Trident RESTful

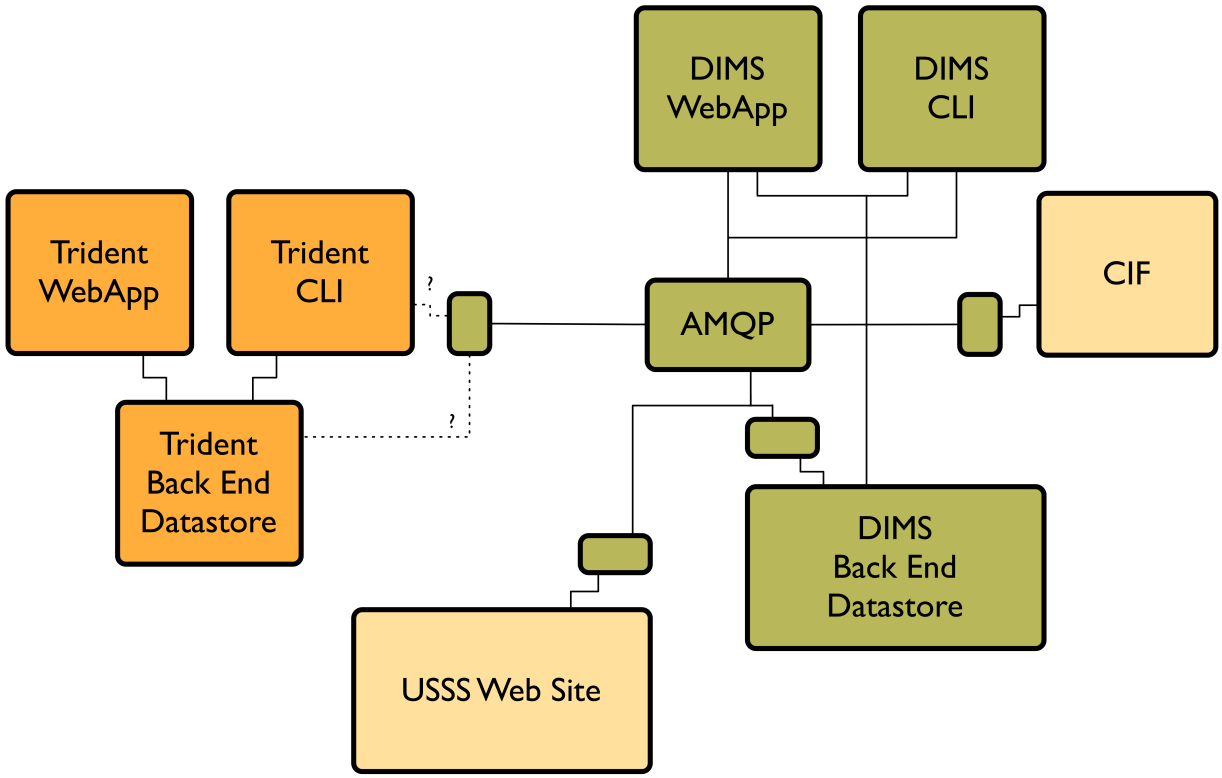


Fig. 5.3: DIMS and Trident Component Interfaces

interface in the same way as the Trident CLI (known as `tcli`, pronounced "tickly"), or they can use the PRISEM remote data query mechanism to front-end `tcli`. (See Figure *DIMS and Trident Component Stack*.) The former is likely the simplest and most robust mechanism for web application GUI-to-backend data flows.

The PRISEM system used an obsolete (past end-of-life) commercial SEIM product that collected logs from participating sites, and forwarded them to a central storage and processing system. This is described in the DIMS Operational Concept Description v 2.9.0, Section PRISEM capabilities, and depicted in this document in Figure *PRISEM Initial Deployment and Flows*.

The data flow used in the more modern *MozDef* system was described in Section *Concept of execution*. MozDef uses Python scripts for enrichment of incoming event logs, optionally received via AMQP (using RabbitMQ) (see MozDef Concept of Operations).

To replace this distributed log collection system with an open source alternative, the features of RabbitMQ known as Federated Queues and Distributed RabbitMQ brokers (specifically, the Shovel plugin), implemented in Docker containers like other DIMS components, can be used. This architecture is depicted in Figure *Proposed DIMS-PISCES Collector Architecture*.

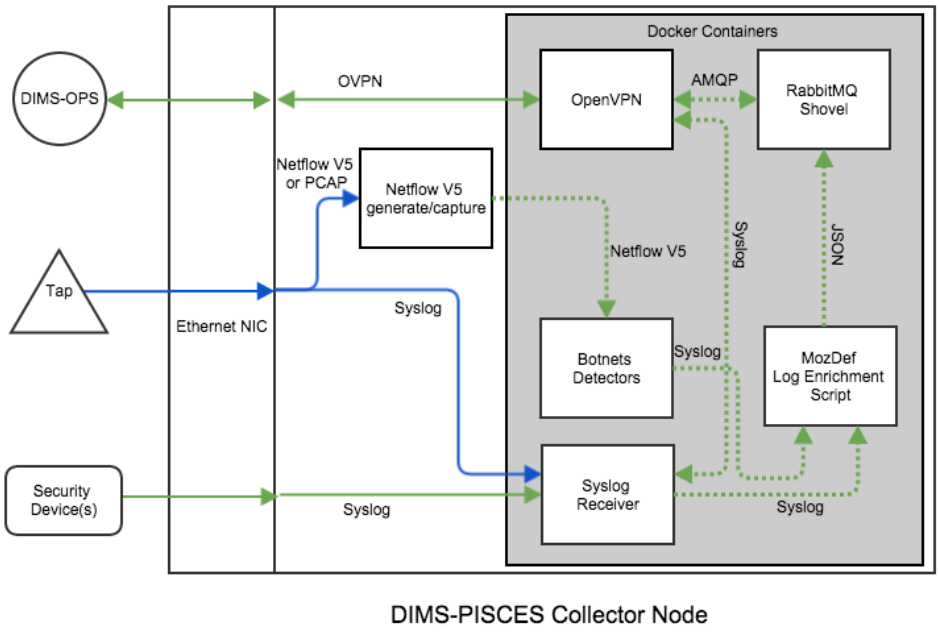


Fig. 5.4: Proposed DIMS-PISCES Collector Architecture

The mechanisms for implementing this distributed collection archicture using RabbitMQ are described in:

- Alvaro Videla - Building a Distributed Data Ingestion System with RabbitMQ, YouTube, Jul 16, 2014
- Distributed log aggregation with RabbitMQ Federation, by Alvaro Videla, December 17, 2013
- Routing Topologies for Performance and Scalability with RabbitMQ, by Helena Edelson, April 1, 2011

As described in Distributed log aggregation with RabbitMQ Federation, the relationship between participant sites with the DIMS-PISCES collector is one of *upstream* exchanges, which will feed the central DIMS-PISCES backend data store acting as a *downstream* exchange via the RabbitMQ Shovel plugin.

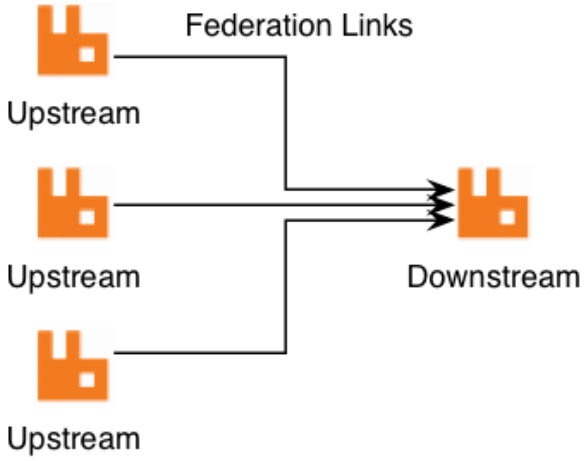


Fig. 5.5: Relationship between Upstream and Downstream Exchanges

Certain types of information that are related to the site where the upstream exchange is located make sense to be included by the producer scripts when queueing events at the upstream for later transport to the downstream exchange. These would be things like geolocation from an off-line database (e.g., Maxmind), and tagging with the SiteID, etc.

Other types of data *do not make sense* to add at the upstream, most notably data that resides at the central backend data store (e.g, data held in the Collective Intelligence Framework (CIF) database, which was described in Section Current system or situation of the DIMS Operational Concept Description v 2.9.0.) In order a producer to tag data using information stored remotely, the producer would have to make a remote query for the data, then insert it, then queue the event log data. This requires that this added data transit the network twice (once in response to the query for it, and again when the event log is transmitted from upstream exchange to downstream exchange.)

It makes more sense to insert a consumer on the downstream exchange that does this enrichment using locally available data, then index it in the backend data store.

Other web pages that provide alternative methods of collecting log events in Docker containers include the following:

- Automating Docker Logging: ElasticSearch, Logstash, Kibana, and Logspout, by Nathan LeClaire, Apr 27, 2015
- Scalable Docker Monitoring with Fluentd, Elasticsearch and Kibana 4, by manu, November 21, 2014
- syslog logging driver for Docker, by Mark Wolfe, May 3, 2015
- Real-time monitoring of Hadoop clusters, by Attila Kanto, October 7, 2014

External Communications Detailed Design

Figure *Conceptual Diagram of Remote VPN Access* shows a conceptual view of remote access to an internal Virtual LAN (VLAN) via an OpenVPN tunnel. Each of the hosts at the top of the diagram (a remote system, such as a data collector node, in the upper left, and two developer laptops at the upper right.)

Remote OpenVPN clients connect to the OpenVPN server and a tunnel interface (`tun0`) is created for each host on the subnet `10.86.86.0/24`. The OpenVPN server provides Network Address Translation (NAT) services to these devices to its internal interface on the internal virtual LAN (`VLAN1`) using the `10.142.29.0/24` network block. Bare-metal and virtual machine servers sharing this VLAN are thus directly accessible behind the firewall.

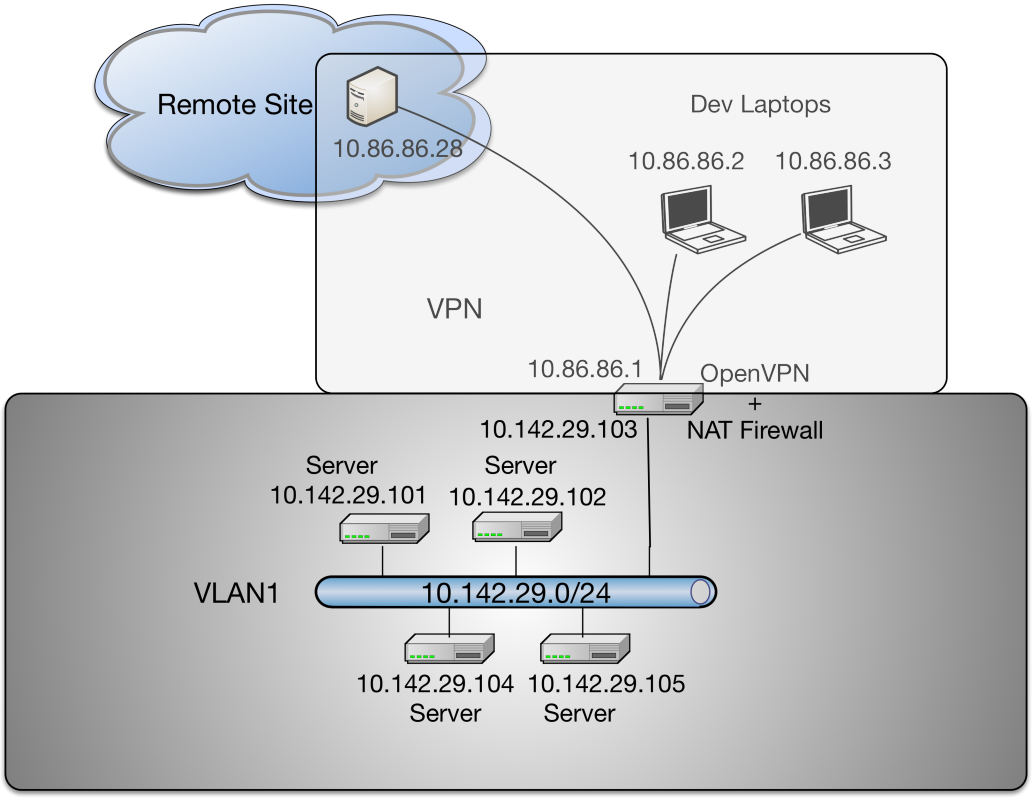


Fig. 5.6: Conceptual Diagram of Remote VPN Access

Note: Not depicted in Figure *Conceptual Diagram of Remote VPN Access* are the specific routable IP addresses that each of the tunnel clients on the top of the diagram, nor the OpenVPn server itself, are using. The OpenVPN server is shown as splitting the two boxed virtual networks to indicate its role in providing remote access that connects the two virtual networks by way of a tunnel using the network address range `10.86.86.0/24` in this case. To include the Internet-routable IP addresses, while being more precise, complicates the diagram. These laptops have two interfaces (one wired, one wireless) that can be used for Internet access required to connect to the OpenVPN server via a public IP address.

Note: To facilitate development and testing of components during the development phase of the DIMS project, multiple OpenVPN tunnels were used to provide relatively unrestricted remote access to internal DIMS systems until the platform stabilized and tighter access control rules applied. The team did not have the staff resources to start out with tight access controls and be able to diagnose problems that could be caused by either service misconfiguration, network routing misconfiguration, DNS misconfiguration, or access control misconfiguration. Thus a more open architecture was used to lessen friction during development across multiple timezones and multiple sites, with many team members also using mobile devices.

In practice, this kind of remote VPN access is only required for development activities that are not easily supported by either SSH tunnels, or SSL connections to AMQP or other specific services. For example, Ansible uses SSH, so configuration management and CI/CD functions do not require a full OpenVPN tunnel. Containerized microservices using Docker networking can use SSL for tunneling, providing their own equivalent of OpenVPN tunnels.

CHAPTER 6

Requirements traceability

The following table is found in Section CSCI capability requirements of DIMS System Requirements v 2.9.0. Each of the elements below links to the relevant section.

CSCI	Label	Contract Item
Backend Data Stores (BDS) CSCI	BDS	C.3.1.1
Dashboard Web Application (DWA) CSCI	DWA	C.3.1.1
Data Integration and User Tools (DIUT) CSCI	DIUT	C.3.1.2
Vertical/Lateral Information Sharing (VLIS) CSCI	VLIS	C.3.1.3

CHAPTER 7

Notes

This document is structured on MIL-STD-498, described at A forgotten military standard that saves weeks of work (by providing free project management templates), by Kristof Kovacs. Specifically, this document is modelled on SSDD.html.

Glossary of Terms

Agile A programming methodology based on short cycles of feature-specific changes and rapid delivery, as opposed to the "Waterfall" model of system development with long requirements definition, specification, design, build, test, acceptance, delivery sequences of steps.

Botnets System The name given to the re-implementation of *Einstein 1* technology. See http://web.archive.org/web/20131115180654/http://www.botnets.org/

cron A Unix/Linux service daemon that is responsible for running background tasks on a scheduled basis.

Git A source code version management system in widespread use.

CIFglue "Simple rails app to quickly add indicators to the Collective Intelligence Framework"

Cryptographic Hash

Cryptographic Hashing Algorithm A mathematical method of uniquely representing a stream of bits with a fixed-length numeric value in a numeric space sufficiently large so as to be infeasible to predictably generate the same hash value for two different files. (Used as an integrity checking mechanism). Commonly used algorithms are MD5, SHA1, SHA224, SHA256, RIPEMD-128. (See also http://en.wikipedia.org/wiki/Cryptographic_hash_function).

I An aggregation of software that satisfies an end use function and is designated for separate configuration management by the acquirer. CSCIs are selected based on tradeoffs among software function, size, host or target computers, developer, support concept, plans for reuse, criticality, interface considerations, need to be separately documented and controlled, and other factors.

Einstein 1 A network flow based behavioral and watchlist based detection system developed by University of Michigan and Merit Networks, Inc. for use by US-CERT. The re-implementation is known as the *Botnets System*.

Fusion Center Entities created by DHS to integrate federal law enforcement and intelligence resources with state and local law enforcement for greater collaboration and information sharing across levels of SLTT governments.

GZIP Gnu ZIP (file compression program)

MUTEX Mutual Exclusion (object or lock, used to synchronize execution of independent threads or processes that must share a common resource in an exclusive manner, or to ensure only one copy of a program is running at a time)

NetFlow Record format developed by Cisco for logging and storing Network Flow information (see also SiLKTools).

NoSQL The term for database that does not use the typical table-based relational schema as Relational Database Management Systems (RDBMS)

Ops-Trust (ops-t) Operational Security Trust organization (see http://ops-trust.net/)

Redis A "NoSQL" database system used to store files in a key/value pair model via a RESTful HTTP/HTTPS interface.

SiLKTools A network flow logging and archiving format and tool set developed by Carnegie Mellon's Software Engineering Institute (in support of CERT/CC).

Team Cymru (Pronounced "COME-ree") – "Team Cymru Research NFP is a specialized Internet security research firm and 501(c)3 non-profit dedicated to making the Internet more secure. Team Cymru helps organizations identify and eradicate problems in their networks, providing insight that improves lives."

Tupelo A host-based forensic system (client and server) developed at the University of Washington, based on the Honeynet Project "Manuka" system.

List of Acronyms

AAA Authentication, Authorization, and Accounting

AMQP Advanced Message Queuing Protocol

AS Autonomous System

ASN Autonomous System Number

CI Critical Infrastructure

CIDR Classless Internet Domain Routing

CIF Collective Intelligence Framework

CIP Critical Infrastructure Protection

CISO Chief Information and Security Officer

COA Course of Action (steps to Respond and Recover)

CONOPS Concept of Operations

CRADA Cooperative Research and Development Agreement

CSIRT Computer Security Incident Response Team

CSV Comma-separated Value (a semi-structured file format)

DIMS Distributed Incident Management System

DNS Domain Name System

DoS Denial of Service

DDoS Distributed Denial of Service

EO Executive Order

HSPD Homeland Security Presidential Directive

ICT Information and Communication Technology

IOC Indicators of Compromise

IP Internet Protocol (TCP and UDP are examples of Internet Protocols)

IRC Internet Relay Chat (an instant messaging system)

JSON JavaScript Object Notation

MAPP Microsoft Active Protections Program

MNS Mission Needs Statement

NCFTA National Cyber-Forensics & Training Alliance

NTP Network Time Protocol (a service exploited to perform reflected/amplified DDoS attacks by spoofing the source address of requests, where the much larger responses flood the victim)

OODA Observe, Orient, Decide, and Act (also known as the "Boyd Cycle")

PPD Presidential Policy Directive

PRISEM Public Regional Information Security Event Management

RBAC Role Based Access Control

RESTful Representational State Transfer web service API

RPC Remote Procedure Call

SCADA Supervisory Control and Data Acquisition

SIEM Security Information Event Management (sometimes referred to as Security Event Information Management, Security Event Monitoring, causing some to pronounce it as "sim-sem".)

SITREP SITuational awareness REPort

SLTT State, Local, Territorial, and Tribal (classification of non-federal government entities)

SOC Security Operations Center

SoD Security on Demand (PRISEM project support vendor)

SSH Secure Shell

STIX Structure Threat Information Expression. A standard for information exchange developed by MITRE in support of DHS US-CERT.

TAXII Trusted Automated Exchange of Indicator Information

TCP Transmission Control Protocol (one of the Internet Protocols)

TLP Traffic Light Protocol

TTP Tools, Tactics, and Procedures

UC Use Case

UDP Unreliable Datagram Protocol (one of the Internet Protocols)

WCX Western Cyber Exchange

CHAPTER 8

License

Section author: Dave Dittrich (@davedittrich) <dittrich @ u.washington.edu>

```
Berkeley Three Clause License
=============================

Copyright (c) 2014 - 2017 University of Washington. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER 9

Contact

Section author: Dave Dittrich <dittrich @ u.washington.edu>

Section author: Stuart Maclean <stuart @ apl.washington.edu>

Index

S

T

U

W