SAVIOR SYSTEM ARCHITECTURE AND DESIGN

Version 2, January 2019

Next Century Team

chris.long@nextcentury.com

# Introduction

This document describes the architecture of the SAVIOR (Secure Applications in Virtual Instantiations of Roles) system, as provided for the Final Evaluation.

Diagrams of the SAVIOR system are shown in **Figure 1** and Figure 2 (on pages 7 and 8, respectively). The components of SAVIOR can be divided into four categories based on where they execute and how many instances of them exist:

* Client-side
* Ecosystem
* Virtue
* User

The components in each category are described below, followed by a description of external services used by SAVIOR.

# Client-side

SAVIOR has three client-side components:

* Desktop
* Admin(istrator) Workbench

The Desktop is a Java application that runs on Windows or Linux computers[[1]](#footnote-1). SAVIOR users authenticate to and connect to the SAVIOR system using the Desktop.

The Desktop connects to other components in SAVIOR as shown in Table 1.

The Admin Workbench is a web application designed to allow a SAVIOR administrator to set up and maintain the configuration of the SAVIOR system. Details of its features and usage are provided in the Administrator Information document.

As web application, the Admin Workbench runs in a web browser and communicates directly only with the Virtue Manager server over http[[2]](#footnote-2).

Table 1 Desktop connections

|  |  |  |
| --- | --- | --- |
| Component | Protocol/Format | Description |
| Virtue Manager | http/json | User authentication and Virtue instance actions (e.g., starting an application) |
| Linux VM (Xen DomU) | ssh | Start/stop XPRA[[3]](#footnote-3) and application launch |
| Linux Applications | XPRA over ssh | Graphical I/O for Linux applications |
| Windows Display Server | ssh | Start/stop XPRA and application launch |
| Windows Applications | XPRA over ssh | Provides a bridge to connect to graphical I/O of Windows applications |

# Ecosystem

The SAVIOR ecosystem consists of the components that run at all times while SAVIOR is operating. They are described in the following subsections.

## Virtue Manager

This component supports the Virtue REST API and provides back-end support for the user-facing SAVIOR components: Desktop and Admin Workbench. It communicates with other components as shown in Table 2.

Table 2 Virtue Manager connections

|  |  |  |
| --- | --- | --- |
| Component | Protocol/Format | Description |
| Desktop | http | Uses Virtue REST API with SAVIOR extensions |
| Admin Workbench | http | Serves web application and uses Virtue REST API with SAVIOR extensions |
| Virtue/User DB | Filesystem | Resident on same host, in the filesystem. |
| Active Directory Service | ldap | Used for user authentication |
| Xen hypervisor | ssh | Initial Xen startup and managing application DomUs |
| Linux application DomUs | ssh | Startup configuration |
| Windows application VMs | ssh | Startup configuration |
| Windows Display Server | ssh | Startup configuration and launch Windows applications |
| CIFS Proxy | ssh | Startup configuration |
|  | http | Configure individual file shares |

## Virtue/User DB

This component stores all metadata associated with users and Virtues, for example:

* Which users can are allowed to run which Virtues
* What applications are supported by each Virtue

The Virtue Manager uses a database persistence layer[[4]](#footnote-4) that makes it database-agnostic. In the current deployment, the Virtue/User DB is H2[[5]](#footnote-5), which runs as a library in its process and uses its local filesystem as the persistent store. It can be configured to use others, such as PostgreSQL on a separate host, with which it has been tested. Communications with such a database are encrypted, for example with SSL for PostgreSQL.

## Sensing & Response Certificate Authority

All communications between Sensors (embedded in Virtues as well as infrastructure components like the Xen Hypervisor and Xen Dom0 controller) and the Sensing & Response infrastructure are secured via mutually authenticated Transport Layer Security (TLS) certificates. The Sensing & Response Certificate Authority (CA) is based on the CloudFlare CFSSL Certificate Authority, a stand-alone CA that provides a custom Root Certificate and individual server and client certificates for all SAVIOR infrastructure. Interactions with the Certificate Authority are managed through the Sensing & Response API, with a Certificate Registration protocol based on the IETF ACME Standard[[6]](#footnote-6).

## Sensing & Response Server

The Sensing & Response Server, or the *Sensing API*, is comprised of a custom control API and a PostGRES Database. The PostGRES Database holds global sensor configuration data, sensor registration data, and system control data used by the Sensing API. The Sensing API is an Elixr/Erlang based RESTful API that handles:

* Sensor to API certificate negotiation, based on the ACME Standard.
* Sensor registration and configuration, based on a modified ACME Standard protocol.
* API to Sensor control and actuation, through mutually authenticated (and certificate pinned) bi-directional RESTful interactions.
* Administrator to API control and inspection of Sensors
* Administrator management of Sensor configurations

## Kafka

Event logs from all Sensors controlled by the Sensing API are streamed to an Apache Kafka cluster, with every instance of a Sensor issued a distinct Kafka *Topic*. Kafka *topics* are generated as randomized UUIDs at Sensor registration time, to prevent easy targeting of specific topics for monitoring or intrusion purposes. The association between *topics* and Sensors is carefully controlled by the Sensing API, with most log streaming interactions handled transparently by the Sensing API directly.

# Virtue

This section describes SAVIOR components that run as part of a Virtue. Some of these components may be instantiated more than once within a single Virtue. However, all are bound to a specific Virtue and none outlive that Virtue or are used by more than one Virtue.

**Figure 1** and **Figure 2** show representative Virtue instantiations. In general, the hierarchy of a SAVIOR Virtue is:

* Xen hypervisor, exactly one per Virtue, on an EC2 instance (currently T3 and C5 instances are supported)
  + Linux Dom0
    - NFS Sensor
    - Firewall
  + Linux application DomUs, zero or more per Virtue
    - One or more applications available, of which zero or more may be running at any given time
    - Xpra server
    - Sensor VM components, always running while the parent DomU is running
  + NFS server unikernel DomU, one per Virtue
  + DHCP server unikernel DomU, one per Virtue
* Windows application VM on an EC2 instance. Zero or more per Virtue, although we expect more than one to be rare
  + One or more applications available, of which zero or more may be running at any given time
  + Sensor VM components, always running while the parent VM is running
* Windows Display Server. Exactly one instance for each Windows application VM in the Virtue.

These components are described below.

## Xen Hypervisor

SAVIOR uses Xen 4.10, which runs unmodified on T3 instances, on which it supports paravirtualized guests. One such instance is used for each Virtue.

In the future we expect that Amazon will update other instance types to allow Xen to run more widely. If SAVIOR needs to run on other instance types before that occurs, for example if a less expensive instance type is needed, Xen Blanket could be used without architectural changes to SAVIOR.

The Virtue Manager communicates with the hypervisor via ssh.

### Dom0

The Xen Dom0 VM runs management services. The Virtue Manager communicates with it via ssh.

#### NFS Sensor

The NFS Sensor monitors, interprets, and reports on traffic to and from the NFS Server Unikernel. It communicates with the Sensing & Response Server over a secure TLS connection.

### NFS Server Unikernel

The NFS Server runs as a unikernel in a Xen DomU. It provides access to storage to both Linux DomUs and Windows application VMs in its Virtue using the NFS protocol.

### Linux Application DomUs

Linux application DomUs host applications accessed by the Desktop. They communicate with the Desktop and Virtue Manager via ssh.

## Windows Application VM

Windows Application VMs host applications accessed by the Desktop through the Windows Display Server. Each one communicates with its own instance of the Windows Display Server via Remote Desktop Protocol (RDP)[[7]](#footnote-7) (using the RDP feature remote applications integrated locally (RAIL)[[8]](#footnote-8)).

## Windows Display Server

The Windows Display Server (WDS) provides a bridge between the Desktop and Windows Application VMs that allows the Desktop to communicate using XRPA and the Windows VM to communicate using RDP. The WDS runs an XPRA server to which the Desktop connects and an RDP client, connected to the Windows Application VM, for each Windows application that is running. The WDS currently exists as a Linux EC2 instance. In the future it may run as a DomU inside the Xen hypervisor, instead.

# User

This section describes SAVIOR components associated with a user and which may be used by more than one of that user’s Virtues.

## CIFS Proxy

The CIFS Proxy provides Virtues access to Windows file servers that do not need to be aware of SAVIOR or Virtues. It receives authentication credentials for a user from the Virtue Manager and uses those credentials to mount Windows file shares. It then exports those file shares to Virtues, with permissions that can vary across Virtues. Which Virtues can access a file share and the permissions are defined in the Admin Workbench.

The CIFS Proxy is controlled by the Virtue Manager via a REST API and is set up and configured with ssh connections (also from the Virtue Manager). It communicates with the Windows file servers and Virtue VMs using Server Message Block (SMB)[[9]](#footnote-9).

# External Services

SAVIOR relies on the following external services:

* Active Directory Server
* DNS server
* DHCP server
* Windows file servers (optional)

In the development and testing environment, SAVIOR has been using Route53 for DNS and DHCP. How these services will be provided in the test environment are TBD.

|  |
| --- |
| **Figure 1** SAVIOR Architecture. Sensor-related connections are shown. |

|  |
| --- |
| **Figure 2** SAVIOR Architecture. User-oriented connections are shown |

1. Some features, such as Single Sign On and clipboard support require platform-specific code. Future versions of SAVIOR may support other systems, such as OS X. [↑](#footnote-ref-1)
2. The certificates required for https are cumbersome for development and testing. For deployment, certificates will be created and https used. [↑](#footnote-ref-2)
3. http://xpra.org [↑](#footnote-ref-3)
4. Java Persistence API (JPA) - https://en.wikipedia.org/wiki/Java\_Persistence\_API [↑](#footnote-ref-4)
5. http://www.h2database.com/html/main.html [↑](#footnote-ref-5)
6. https://tools.ietf.org/html/draft-ietf-acme-acme-11 [↑](#footnote-ref-6)
7. https://msdn.microsoft.com/en-us/library/cc240445/ [↑](#footnote-ref-7)
8. https://msdn.microsoft.com/en-us/library/cc242568/ [↑](#footnote-ref-8)
9. <https://en.wikipedia.org/wiki/Server_Message_Block>, also known as Common Internet File System (CIFS) [↑](#footnote-ref-9)