Michigan State University

Team Rook

Anomaly Detection Suite v2.0

Project Plan

Fall 2016

Rook Security Staff:

Bob Dyksen

Mat Gangwer

Michael Taylor

J.J. Thompson

Vigo Wei

Michigan State University Capstone Members:

Cam Gibson

Brian Harazim

Grant Levene

Zach Rosenthal

Andrew Werner

Table of Contents

Executive Summary

Commented [1]: I think this section is done. Maybe just some proofreading

Rook Security is a Managed Security Services Provider that manages, anticipates, and eliminates threats to their client's electronic data and property. Based out of Indianapolis, Indiana, Rook specializes in leveraging state-of-the-art technology, a team of professionals, and cross-functional research to provide solutions to their clients that decrease negative outcomes, give accountability, and maximize business value.

Among Rook's solutions is the Anomaly Detection Suite (ADS) v1.0, an appliance which is added to a client's network in order to protect the network against a wide range of cybersecurity threats. After installation, the appliance then receives and analyzes all network communications utilizing Rook's patent-pending algorithm. However, the recent advent of large-scale virtual computing has created a need for a different approach in supporting clients that have networks with many virtual environments. Additionally, in their mission to use and account for the latest technologies, Rook is looking to add new techniques and optimizations to the Anomaly Detection Suite v1.0.

The Anomaly Detection Suite v2.0 addresses these issues of virtual environments by deploying highly specialized agents running on all of the client's computers, including both Windows and Linux operating systems. In addition to Rook's patent-pending algorithm, a machine learning component is integrated into the analysis of the network communication. Rather than relying on predetermined statistical thresholds, incorporating machine learning allows the analysis portion of ADS to learn from the abundant amount of data it handles and use this information to make more calculated decisions.

ADS v2.0 includes a new web-based management dashboard that provides real-time visual representations of detected anomalies, threat statistics, and information regarding agent health. The dashboard enables admins to deploy and configure agents remotely. It also ensures analysts can quickly find and act upon infringing anomalies, as well as ensure all agents are working properly. Overall this allows for a more efficient way for a client to manage all the agents in the client's network.

Functional Specification

Anomaly Detection System v2.0

The goal of this project is to improve the ADS v1.0 by incorporating machine learning to increase accuracy, creating a web interface that provides updates, statistics, and status for all the agents in a client's network at a quick glance, optimizing the analysis engine to be less CPU intensive, and creating a Linux version of the agent using the Windows version as a template. With the addition of these improvements, the original suite becomes ADS v2.0.

Machine Learning Engine

Machine learning is integrated into the analysis engine of ADS v2.0 in order to increase the accuracy. Previously, ADS v1.0 performed statistical analysis on the network traffic to determine any anomalies based on thresholds set in advance. However, with any anomaly detection system, there is a chance that network traffic may be incorrectly classified as an anomaly. Such an event is called a false positive and it is the goal of ADS v2.0 to classify as many anomalies correctly while minimizing the number of false positives. Once anomalies are detected, the information is relayed to the web management dashboard.

In recent years, anomaly detection systems have utilized machine learning to improve performance. The machine learning engine is created in such a way that it is able to utilize a variety of algorithms, such as clustering, linear regression, and decision trees. In this way, Rook is able to test the effectiveness of a particular algorithm and decide which would be optimal for the ADS.

Web Management Dashboard

A major part of this project is the management dashboard. The management dashboard has a visual of the state of each agent (CPU usage, memory, disk usage), notifications of alerts on an agent, history of the alerts, and other statistics. Not only does this act as a nice place to keep track of all the agents in a network for a client, but it allows for the agents connected to the dashboard to be remotely updated. The dashboard allows the user to select what version an agent should be updated to when it checks into the dashboard next. This is

helpful if Rook wants to test an update for a group of agents at a specific location because it can be done all in one place remotely.

The management dashboard will be given to the client to allow a system manager to keep track of what is happening in their network. An important part of the management dashboard is the user interface because if the user interface is difficult to use, then the client will likely not use it.

The management dashboard provides a convenient way to keep track of the agents on a client's network. Previously the agents were being monitored by a text space user interface and thus there was no easy way to keep track of the agents in the network at one location, along with agents in a network at a different location. The management dashboard helps clients who have multiple locations by providing a unified location to monitor the status of multiple networks from different locations.

Optimizations

One of the goals of ADS v2.0 is to improve the efficiency of ADS v1.0. The suite must retrieve, process and store a lot of data to function properly. Interacting with the databases creates a bottleneck, since this is much slower than the time it takes the CPU to do the processing. We used GNU gprof and OProfile to profile the suite and find the exact bottleneck locations. The profiles agreed with our hypothesis that database accesses were killing the suite's performance.

To help mitigate this, the suite makes use of an Extract, Transform and Load (ETL) paradigm, a process with three components. The extraction component is responsible for retrieving data. The transformation component is responsible for the data manipulation or processing. Finally the load component is responsible for storing the data. These three components happen in an asynchronous fashion, meaning as soon as data moves through one component the next component immediately starts to handle that data. ETL not only helps to lessen the footprint of the database accesses, but also helps to organize the flow of data. ETL is only mildly successful in improving the overall performance of the suite.

To further the improvement in ADS v2.0, we have outlined a path to unloading some analysis from the server onto the individual agents. This will help to greatly improve performance as the server will no longer be responsible for handling the analysis of the client's agents. In this scenario, each agent will be responsible for its own analysis.

Linux Agent

A majority of current operating systems are on some version of Windows. The Windows version of the detection suite is already being used by Rook's clients. The detection suite is used for clients who do not have the ability to use an appliance on their network, so it is done virtually. Creating a Linux based agent allows ADS to be available for the untouched part of the market. The detection suite acts the same as the one on Windows and has all the same capabilities. These capabilities are the improvements that were made to the Windows agent for performance and optimization, as well as encrypting all communication between the agents and servers. Having ADS v2.0 available on these operating systems allows Rook to access clients that they normally wouldn't.

Design Specification

The web management dashboard is accessed by client network administrators to monitor, update, and interact with remote agents. The dashboard gives users an easy way to get an overview of their network's health, filter through network agents, or manage them on an individual basis. The user can search through the table of agents or filter by a certain column in order to better understand the status their network is in.

Home Page

Upon opening, users will be presented with info and graphics summarizing the suite's current outlook on the network. Configurable measures might include engine CPU usage, number of agents online, number of anomalies that day, etc. Additionally, users will have a notifications area to see what some of the more recent and significant network changes are. Also, any agents that have not checked in on schedule will be visible to the user.

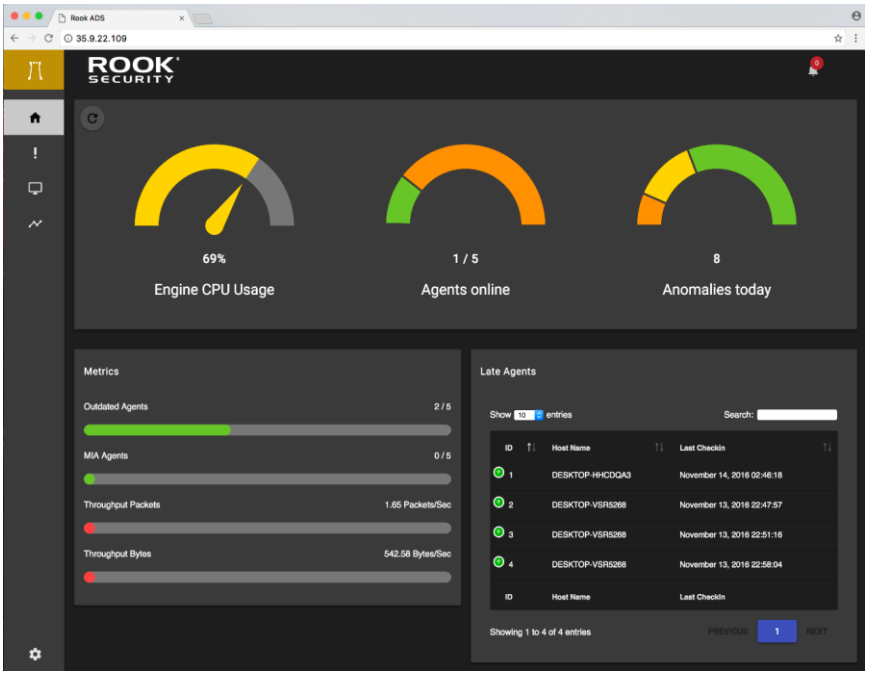


Figure 1: Home Page

Alert Page

The alert page is a location for a system administrator to view all detected anomalies. The information is displayed in a table with information about the IP source and destination, time of occurrence, type of anomaly, name given by ADS v1.0, and a score. The score is determined by ADS v1.0 and gives an idea of how persistent and dangerous the anomaly is.

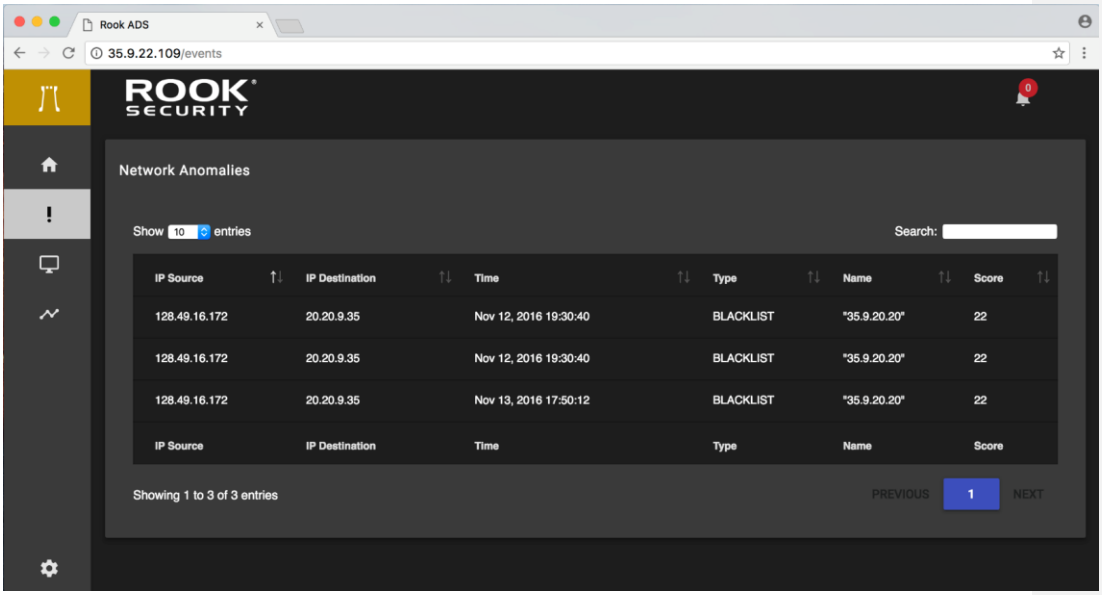


Figure 2: Alert Page

Agents Page

To allow agent administration on both a group and individual basis, users will be presented with a filterable and sortable table filled with the network's agents. Each row contains the most relevant agent properties, such as host name, operating system, last check-in time, CPU usage, status, agent version, etc. Agents can be filtered by any of these properties via clicking on the column name. Searching through the agent table for a keyword is available through the search box.

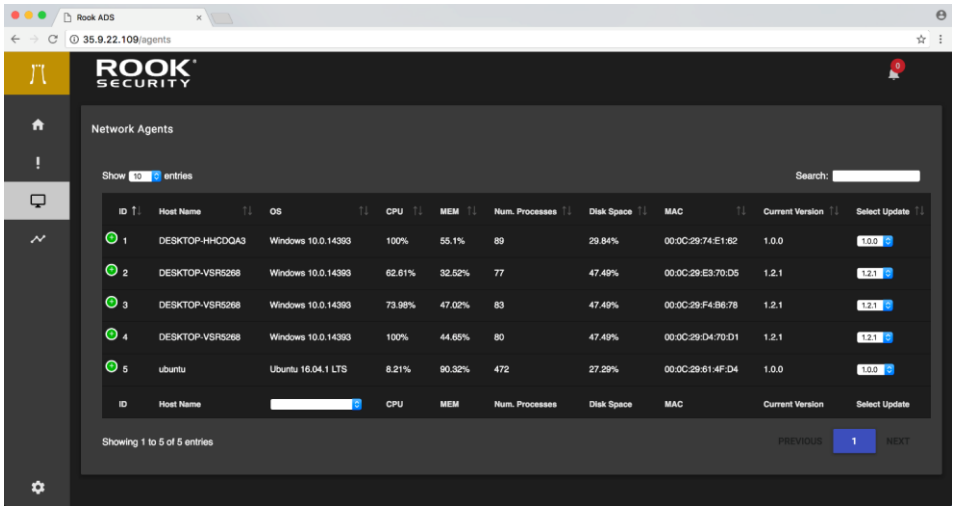


Figure 3: Agent Table

Additionally, clicking on an agent opens more information for that agent, as seen in Figure 4. From here, agents can be selected to notify that they need an update.

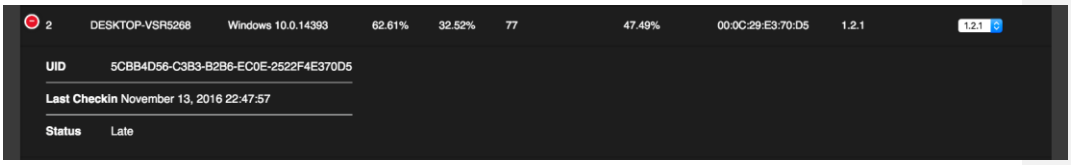


Figure 4: Selected Agent

Reporting Page

In order to facilitate greater ease of visualization and management, a reporting page will be a great asset. Administrators can filter and compare agents by different properties or events with the assistance of charts and reports.

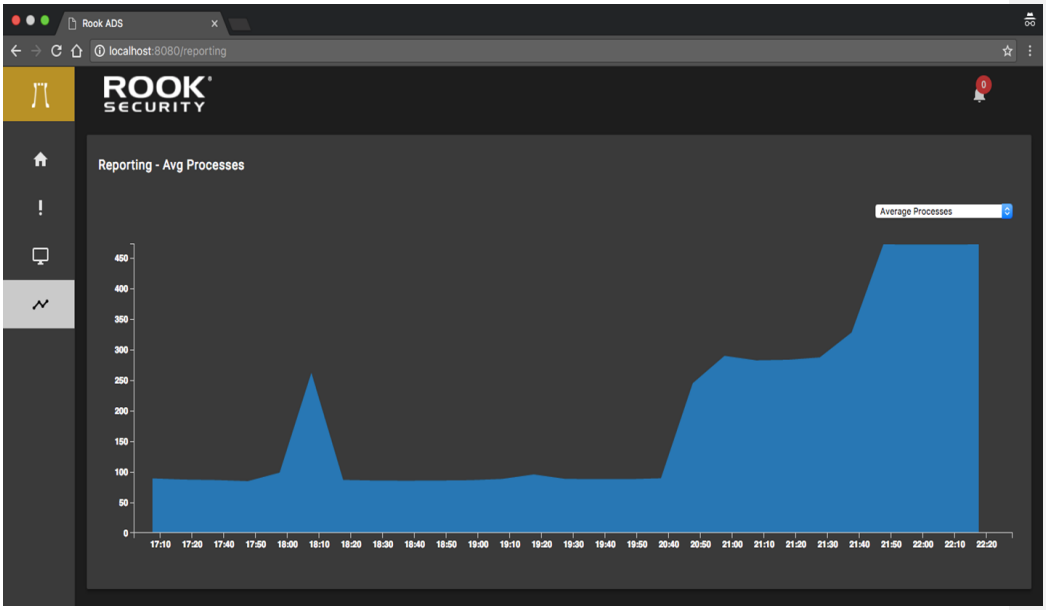


Figure 5: Reporting Page

Technical Specification

Software Technologies

- Ubuntu
- Docker / Docker Compose
- C
- Python (Django)
- Node.js
- ReactJS
- HTML / CSS
- OpenSSL
- Redux

Development Environment

- Pycharm
- CLion
- Visual Studio
- Node.js / NPM
- GIT VCS (Bitbucket)
- Postman

System Architecture

Administrators interact with agents and the analysis engine via a web application. The web application serves as a management dashboard making health statistics and alerts available. The health statistics include data throughput, disk utilization, memory utilization, number of anomalies, and severity of anomalies. We leverage the Django Rest Framework for our backend and ReactJS for our frontend. Our Django API stores data in a MySQL database. We created a component with Node.js to send alerts and usage data from the analysis engine to our Django backend.

The main components of the management dashboard are our Django API and our ReactJS app. The Django API exposes endpoints allowing our ReactJS app to access and update all of the necessary data. To support updating the agents in a secure way, the agents “check in” with the server at certain configurable time intervals. During this check in the agents report their current health and our backend checks if the agent needs to be updated. If the agent does need to be updated, the response contains the new desired version of the agent. The agent pulls the required files to update from the server. This allows the agent to update securely since the dashboard cannot push an update.

All of our components are containerized into separate Docker containers. Using Docker allows us to separate the various pieces of our system into smaller more maintainable parts. Docker Compose makes shipping the Anomaly Detection Suite to clients seamless and also enables the system to run in almost any environment.

Alongside the Web Application Server is the ADS server, which is based off of the existing ADS v1.0 architecture. Agents send their network traffic to the ADS, which focuses on the packet headers and not the actual payload of the traffic. It replays the headers to the capture engine, while storing a list of all agents seen. The capture engine extracts the desired information and stores it into a MySQL database. From here, the analysis engine, running Rook's patent-pending algorithm, sifts through the data and detects possible alerts. This information is sent to the machine learning engine in order to more precisely identify true anomalies, which are inserted into the SQLite database. Lastly, the data relay engine serves as a way to transfer information from the detection suite to the backend of the web management dashboard. The communication and structure of our architecture is illustrated in Figure 5.

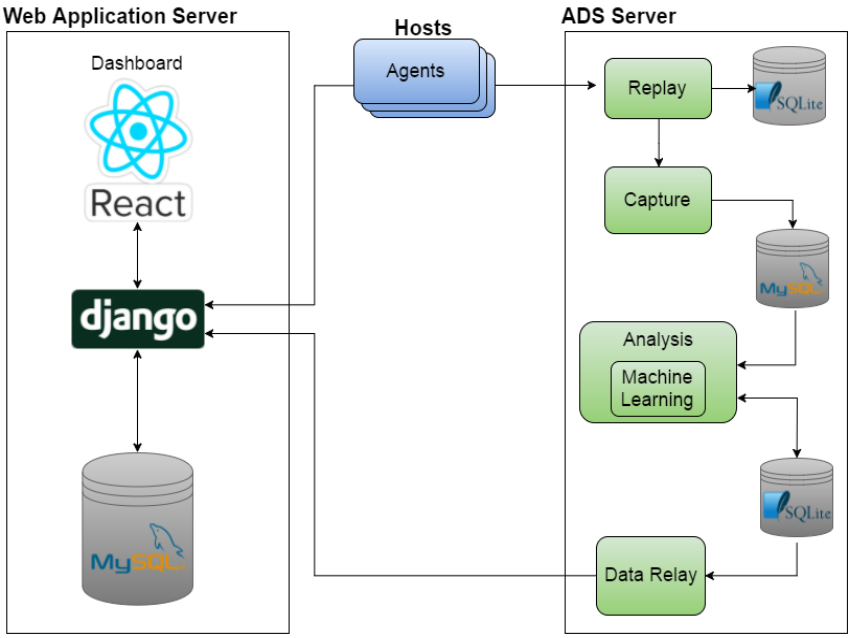


Figure 6: System Architecture

Machine Learning

The analysis engine is the core of the Anomaly Detection System; it analyzes network traffic to identify abnormal network traffic. The analysis engine is written in C. The analysis engine previously used only statistical analysis to determine anomalies. We added a configurable component that allows a machine learning algorithm to be implemented into the analysis engine to better detect anomalies.

Along with the wrapper component, we have added support for the K-Nearest Neighbor (KNN) algorithm, based on distances and density, and the K-Means Clustering algorithm. The machine learning component gathers features from the available network traffic such as source IP address, destination IP address, time of occurrence, score (determined by Rook's analysis algorithm), source port, destination port, number of packets, and number of bytes. For the KNN algorithm, distances are calculated as a function between a new network event and the normal network traffic, in order to classify anomalous data and alert the system administrators. Most importantly, the machine learning algorithms can be tuned in order to maximize the percentage of true positive alerts and minimize the percentage of false positives.

Django API

Show Agents – Returns a listing of all agents
GET: /agent2
Show Agent – Returns details on a specific agent
GET: /agent2/:id
URL Params: id=[string]

Create Agent – Registers a new agent
POST: /agent
Body Params:
Required:
macdata=[string]
iddata=[string]
hostdata=[string]
osdata=[string]
cpudata=[float]
memorydata=[float]
bytesAvaildata=[integer]

bytesTotaldata=[integer]
processesdata=[float]
versiondata=[string]

Agent Check-in – Report data for an existing agent
POST: /agent/:id
URL Params: id=[integer]
Body Params:
Required:
macdata=[string]
Optional:
iddata=[string]
hostdata=[string]
osdata=[string]
cpudata=[float]
memorydata=[float]
bytesAvaildata=[integer]
bytesTotaldata=[integer]
processesdata=[float]
versiondata=[string]

Agent Check Version - called by agents to check if update is available, new version is returned
POST: /update
Body Params:
Required:
id=[string]

Update Agent Version - set the version of an agent
POST: /version
Body Params:
Required:
id=[string]
version=[string]

Show Overview - returns usage statistics, agent statistics, and alert statistics
GET: /home

Body Params:

User Alerts - retrieves alerts not yet viewed by a certain user (alert notifications)
GET: /log
Body Params:
Required:
    id=[integer]

Show Alerts - retrieve all alerts
GET: /alert
Report Alert - called from ADS data relay component to report a new alert
POST: /alert
Required:
    alerts=[array of alerts]
Note: an alert is of the form
{ "ipSrc": integer, "ipDst": integer, "type": integer, "name": string, "time": integer, "score": integer }

Show Data Throughput - retrieve the data throughput for the last hour
GET: /throughput

Report Data Throughput - called from ADS data relay component to report throughput
POST: /throughput
Required:
    throughput=[hour: integer, secs: integer, bytesThrough: integer, packetsThrough: integer]

Show Graph Data – retrieve data for showing graphs
GET: /graph/:id
Required;
    id=[integer]

Notification Callback – called after a user has viewed notification
GET: /notification/:id
Required:
    Id=[integer]

Risk Analysis

Limited knowledge of technologies

We didn't have much experience using Django, React, and network programming. In order to mitigate this, we created basic prototypes and skeletons in order to get comfortable and gain some knowledge with using them. This was mainly done by following tutorials online. We also didn't have any experience in Windows development so we did research into what a Windows application needs.

Getting traffic for testing

To correctly simulate what packets a client might have going through the agent, we needed to produce enough relevant traffic in order have accurate tests. To simulate normal traffic we used Wireshark to capture packets of normal system usage and replayed these packets. To simulate attacks, we again used a replay application to simulate publicly available captured files consisting of network attacks. We also used Metasploit and commands on a terminal such as ping and secure copy to produce an anomaly.

Writing secure code and keeping software secure

Secure code is always important for all projects, but since we are representing a security company we made sure to keep our servers protected and the company software safe. We learned how to write secure code in regards to the agent and the analysis engine. In order to keep Rook's software safe, we did not leave our computers unlocked, and we encrypted our capstone lab and personal hard drives. We also limited the access to our servers by configuring which IP addresses are able to gain entry to the servers.

Machine learning and getting a baseline dataset

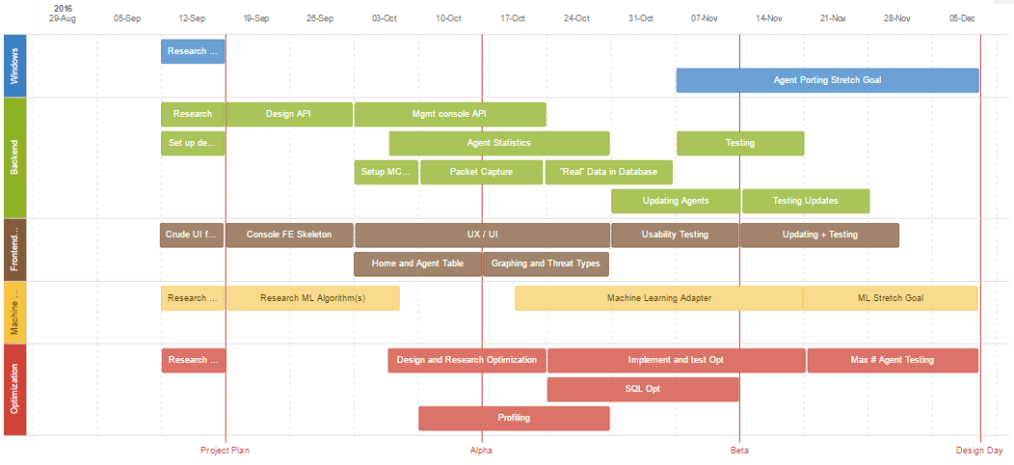
We did not have any prior knowledge about machine learning, so we needed to do some research on its capabilities and how to approach integrating it into ADS v1.0. This required a knowledge of different types of network traffic, a deep understanding of the ADS v1.0 implementation, and a good baseline dataset for the machine learning algorithms to "learn" from. We looked into some open source libraries to get an idea of machine learning resources along with studies of machine learning for network analysis. We also intently studied the code for ADS v1.0 to learn about how to incorporate machine learning. Lastly, we used the existing alerts created from ADS v1.0 as our training data for the algorithm to learn from.

Commented [2]: I switched the first risk to past tense. does that sound good or should it be in future as the others are?

Commented [3]: I think they should all be past tense. No?

Commented [4]: agreed

Timeline



Week 1 (8/31 - 9/3)

- Meet with team
- Schedule conference call
- Research computer networks and security

Week 2 (9/4 - 9/10)

- Set up servers
- Meet with Bob and Taylor from Rook
- Start Project Plan and Status Report

Week 3 (9/11 - 9/17)

- Status Report presentation
- Finish Project Plan
- Research machine learning
- Setup development environment for the Windows agent
- Finish setting up and hardening servers

Week 4 (9/18 - 9/24)

- Project Plan presentation
- Create skeleton for management dashboard
- Start creating management dashboard backend API
- Work on basic machine learning

Week 5 (9/25 - 10/1)

- Connecting fake agent data to dashboard
- Set up dashboard libraries
- Research applicable machine learning algorithms

Week 6 (10/2 - 10/8)

- Add endpoints for the agent table on the dashboard
- Gauge and other visuals created for dashboard
- Created Design Day Booklet page
- Analyze ADS v1.0 engine code for optimization
- Work on flow of management dashboard

Week 7 (10/9 - 10/15)

- Create components for dashboard to fetch data
- Add pie chart and other visuals to dashboard
- Performance testing of engine code
- Prepare for Alpha Presentation

Week 8 (10/16 - 10/22)

- Alpha Presentation
- Agent sends host statistics to dashboard and check in to backend
- Notifications for alerts to dashboard as well as visual improvements
- Look into batch processing for SQL calls for optimization
- Create skeleton for machine learning

Week 9 (10/23 - 10/29)

- Work on updating agents and UI for updating
- Finalized anomaly notification system
- Have real data sent to dashboard
- Decide features for machine learning and prepare data to be sent

Week 10 (10/30 - 11/5)

- Added Redux to dashboard to manage state
- Implemented K-nearest neighbor algorithm
- Begin work on Linux version of agent

Week 11 (11/6 - 11/12)

- Continue work on Linux version of agent
- Added graphs to dashboard
- Prepare for Beta Presentation
- Integrate dashboard with alerts and machine learning
- Update Project Plan

Week 12 (11/13 - 11/19)

- Finishing touches on Windows and Linux version of agent
- Script for project video
- Testing ADS

Week 13 (11/20 - 11/26)

- Beta Presentation
- Work on project video
- Testing ADS

Week 14 (11/27 - 12/3)

- Final testing of all aspects
- Prepare for Design Day

Week 15 (12/4 - 12/10)

- Project Video due
- All deliverables due
- Design Day setup
- Design Day