Project Requirements Document

Cyberthreat Insight Discovery & Visualization Tool

Webroot

Prepared by CU Boulder Capstone Team

November 10, 2019

1. Introduction

Purpose

This is a Requirements Specification document for a new graph interface that will allow Webroot researchers and engineers the ability to construct graphical visualizations of threat content based on Indicators of Compromise (IoC). Webroot is a leading cybersecurity and data protection company that provides its customers with protection from cyber threats. This new system will allow their researchers and engineers the ability to generate visual representations from data generated by information from Webroot's threat assessment modules. Interaction with the interface will allow for specific artifacts querying and be able to extend search parameters by adding more queries to the graph interface. The researchers will also be able to save and reload graphs they create as well as the ability to see and interact with other researches graphs. All of the created graphs will have the ability to view all of the element's IoC details.

This document describes the scope, objectives, and goals of the new system. In addition to describing the project approach, this document contains the high-level requirements, as well as models, use cases and has the stack diagrams. This document is intended to direct the design and implementation of the target system in the given stack architecture.

Responsibilities

The primary responsibilities of the new system:

- should be able to query Webroot's database
- graph node should be able to be moved and interacted with
- provide expandable and dynamic graphs
- customizable graphs to specific users
- allow graphs to be saved
- allow access to all graphs created by researchers
- allow graph locations to be saved per unique graph
- researchers will have to ability to add note or details to the graph

Other desired features of the new system:

- a consistent "look and feel" throughout the interface
- nodes should be selectable and interactable
- nodes should display data about itself
- the graph should be stored on a database
- contents on the information database should not be able to be modified

The system will not be responsible for generating any of the data it interacts with or anything Webroot and this team have determined to be out of scope. This document contains all of the software requirement specifics. It contains a general description of the types of users who will be using our application, how it is going to work, and what technologies we are using to make it work. We will also outline and describe the specific components of the project in the next iteration.

2. Objectives

Business Objectives

- Create an internal tool that Webroot engineers and researchers can use to track threats and inform decisions about future defense modules.

High Level Requirements

- Nodes in the graph will be represented abstractly due to the variable nature of IoCs which are inherent to a document, or a document's history. Connections between IoCs can also be generated from executing binaries. As of the writing of this document, the visualization tool will be capable of representing IoCs including the following list:
    - File hash checksums
    - Fuzzy hash similarity
    - Registry entries
    - Created/Written files & locations
    - Hosts contacted
        - IP / Port
        - DNS
    - File string content
    - similarities between behaviors of binary files

- The graphing interface will have the following main features
    - Ability to query Webroot's DocumentDB
    - Ability for users to physically adjust position / orientation of a graph nodes
    - Ability for nodes to hold hidden connections that can be shown by user choice
        - Implies an ability for nodes to hold references to all possible connections
    - Ability for the graph to change dynamically as new nodes and connections are added by the user
    - Ability for users to create new connections between nodes
    - Ability for users to annotate graphs with (non graph connection) notes
    - Ability to save and load a graph with all user changes made
    - Ability for a user to open a saved graph from another user

- Data in the form of IoCs will be stored in a DynamoDB database managed by Webroot. New data entries into this database trigger a sequence of events that result in autogenerated final tables filled out by Webroot's modules. Our tool will query only these final tables.
- The structure of these tables is subject to change. As such, our tool will have a modular implementation that allows the code to be easily maintained to account for changes in database structure without altering the front end graphical application.

Constraints

- Webroot's current database service through Amazon Web Services (AWS) charges per query. Our design will minimize the number of queries made to Webroot's database by storing data for graph visualization and loading in a separate database. At this time, the separate database will also be an AWS database.
- To illuminate the scalability of the tool, our team will provide a cost analysis based on the number of queries generated and pricing models for various database solutions available from Amazon.

3. Scope

Project Scope

The goal of this project will be to create a web based user interface that displays a graph consisting of many nodes with visual links to show relationships between nodes. The interface will mimic much of the functionality and workflow of Webroot's current too, VirusTotal.

In Scope

- The graph will be the central part of the interface and must have the following attributes:
    - The graph must be expandable and dynamic. That is, there should be some way to add additional nodes and relationships to an existing graph.
    - The orientation of nodes in the graph must be customizable. Users should be able to rearrange the positions of nodes in the graph. The positions of the nodes should also save between sessions, so that if a user returns and loads a saved graph the nodes will appear in the positions where they were last.
    - The graph must be savable and shareable between users. While managing user authentication and accounts is **NOT** within the scope of this project, users should be able to save a graph and load that graph when returning in a new session.
    - Users must be able to annotate the graph in some form. Annotations must save along with the graph state.
- Each node in the graph will have the following capabilities:
    - Nodes must be abstract and extendable. Nodes should be implemented with the assumption that the possible types of nodes will be expanded in the future, and any implementation should allow the addition of node types without significant changes to the source code structure.
    - Nodes should be selectable and display additional information about the object represented by the node. This information will be gathered from the database.
- Part of the project will be to create a database that receives data from the Webroot data pipeline and store that information in a way that is accessible by the graph interface. The database implementation must have the following attributes:
    - The database must rely on an AWS database technology. Any database implementation should be deployable on Webroot's AWS stack and be reasonably cost effective. DynamoDB will be used as a baseline to evaluate any cost difference.
    - Users should not be able to modify the contents of the database. All node information should come directly from the Webroot pipeline, and should only be modified through that pipeline.

- In terms of stack architecture, the full stack will be in scope except for hosting the final product (on servers, or integration with Webroot's current stack), and management of Webroot's current DynamoDB which generates data.

Out of Scope

- Anything that extends beyond the agreed upon project scope and technologies should be assumed to be out-of-scope; though this does not entail that no propositions or technologies outside of the scope can be used.
- Assumptions**:** The project may eventually include a suite of features that we do not currently plan to integrate:
    - We will enable multiple users to share saved graphs. A graph here refers to an annotated, or user-expanded version of the graphs generated from basic queries. As such, we will design the tool to allow multiple users, and to save/load graphs based on a user key. However, it is out of the scope of this project to handle user-authentication and management.
    - We will not currently plan to host or deploy the final version of the product. Dockerizing the tool is within the scope of the project, however.

4. Project Approach

General Approach – Solution Delivery Process

- The CU development team will use an Agile workflow approach
- Each sprint will work toward the idealized "Schedule" listed in the Project Plan section below
- The Agile workflow will allow minor adjustments to be made during throughout the project
- Weekly project status reports will be made available to keep everyone apprised of the project status
- Any proposed major alterations to the development cycle should first be reviewed by the development team, then approved by Webroot and vice versa

Assumptions

- Once the project is delivered, Webroot will perform all required upkeep and maintenance
- The development stack is left up to the CU development team
- The database that receives data from the Webroot data pipeline and store that information in a way that is accessible by the graph interface will rely on AWS technology

Project Risks and Issues

- AWS costs can vary depending on final implementation of pipeline and how queries are implemented
- The project needs to allow for possible changes with the DynamoDB database managed by Webroot that is still in development
- Threat Analysis Pipeline database is new technology that is still undergoing changes

Project Changes

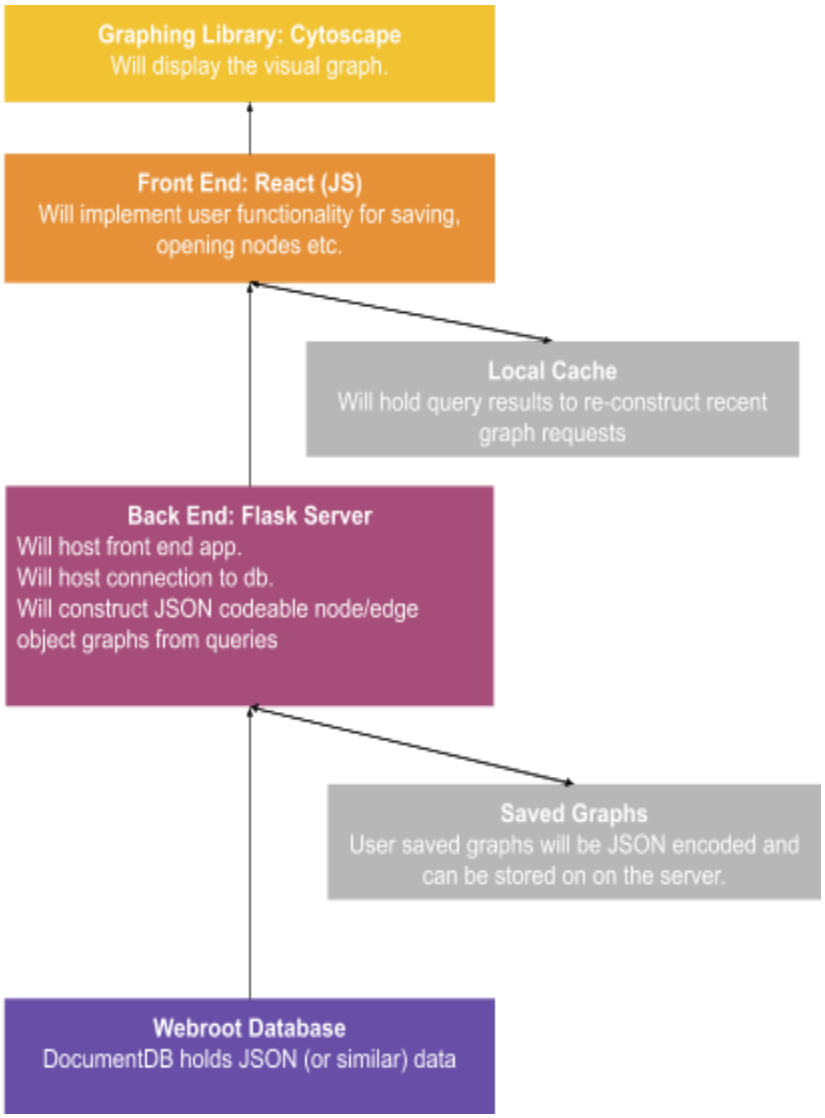
- Changes to project scope and requirements will be reviewed by the requirements lead, Warren Payne. Warren will initiate a discussion with the project team regarding the potential impact of the change on schedule and resources, then provide a decision for approval/disapproval. All scope/requirements changes will be documented and signed off on by both parties.

5. Project Plan

Development Stack

The following ground-up approach provides a flexible framework. We will leverage as many open source tools as possible, such as Cytoscape for front end graphing. At the same time, we propose to use tools that are common to most developers to ensure the simplicity and longevity of the project.

Stack Diagram:



6. The Use Case Model

Use Case Name:	Login User
Summary:	In order for a client to make use of the graphing utility, they must be logged into an account.
Basic Flow:	1. The use case begins when a client indicates that they want to log in. 2. The system requests the username and password. 3. The user enters his username and password. 4. The system verifies the username and password against all registered users. 5. The system begins the login session and redirects the user to either their saved graph or a fresh canvas.
Alternative Flows:	Step 4: if username or password is invalid, the use case goes back to step 2 and the system displays a message indicating an incorrect field.
Extension Points:	none
Preconditions:	The user is registered.
Postconditions:	The client is now able to use the graphing utility.

Use Case Name:	Register User
Summary:	In order to use the graphing tool, the client must be registered. If they have no active account, then they must register with a username and password.
Basic Flow:	1. The use case starts when a user indicates that he wants to register. 2. The system requests a username and password. 3. The user enters a username and password. 4. The system checks that the username does not duplicate any existing registered usernames. 5. The system starts a login session and is redirected to a blank graphing canvas.
Alternative Flows:	● Step 4: If the username duplicates an existing username the system displays a message and the use case goes back to step 2. ● Step 4: If a required field is not filled by the user, the system will display a message indicating the missing field and return to step 2.
Extension Points:	none
Preconditions:	none
Postconditions:	The user is now ready to use the graphing utility

Use Case Name:	Query Webroot Database
Summary:	In order to generate a graph, the user must query the Webroot Database for nodes and specify the desired connections to be displayed.
Basic Flow:	1. The use case begins when the user decides to run a query on the Webroot database 2. The user navigates to the “Search” tab. 3. The system displays an input field prompting for a query input. 4. The user inputs the desired artifact (node) and desired connections. 5. The user selects the “Go” button, which runs the query 6. The system displays the node and the desired connections as a graph network. 7. The user is then free to interact with the graph or to append more queries to the graph.
Alternative Flows:	● Step 6: If the system is unable to fetch any nodes or connections from the given query, a message is displayed indicating that there were no matches found, and the use case returns to step 3.
Extension Points:	*Query Webroot Database, Interact With Graph, Save Graph, Share Graph*
Preconditions:	The user is registered
Postconditions:	The user now has a graph visualization. The user is also now free to interact with the graph or to append more queries to the graph.

Use Case Name:	Interact With Graph
Summary:	With a graph displayed, the user is now able to perform one of the graphing utilities core features: interact dynamically.
Basic Flow:	1. The use case starts when the user selects a node or a connection (edge). 2. The user can drag the node to anywhere within the graphing window. 3. The system adjusts the node's corresponding edges continuously as the node is drug, until it is dropped. 4. The user drops the node where they please.
Alternative Flows:	● Step 1: If the user opts to not drag, but rather just select a node, then the system will produce a window which will display that node's details. The user is then free to select other nodes or exit from the window.
Extension Points:	*Query Webroot Database, Interact With Graph, Save Graph, Share Graph*
Preconditions:	The user is registered, the user has successfully queried the graph (i.e. has a graph to interact with)
Postconditions:	The user no has either a modified graph or has a window displaying a specific node's details.

Use Case Name:	Save Graph
Summary:	If a user intends to exit the graphing utility, then they may elect to save their current graph (queries and graph configuration). This use case describes the process for saving a graph.
Basic Flow:	1. The use case starts when a user decides to save their current graph. 2. The user selects a save icon towards the top of the screen. 3. The system asks for a name for the graph and displays an input field, a *save* button, and a *cancel* button. 4. The user inputs a graph name and selects *save* 5. The system saves the graph
Alternative Flows:	● Step 4: the user may opt to *cancel*. The system then exits the saving window without saving.
Extension Points:	*none*
Preconditions:	The user is registered, the user has successfully queried the graph (i.e. has a graph to interact with)
Postconditions:	The graph is now saved for later use

Use Case Name:	Share Graph
Summary:	The user can opt to share a graph with one or many other users. This use case describes the process of sharing a graph
Basic Flow:	1. The use case starts when the user selects the *share* button towards the top of the page. 2. The system opens a window that prompts for the username of the user that the graph will be shared with, a *share* button and a *cancel* button. 3. The user inputs the username of the user to share the graph with and selects *share*. 4. The system checks for the existence of the inputted username. 5. The system shares the graph with the other user, displays a *sent* confirmation, and exits the sharing window.
Alternative Flows:	● Step 2: the user may opt to *cancel*. The system then exits the sharing window without sharing. ● Step 4: the system may not find a matching username in the database, in which case the system displays a message indicating that the username inputted is invalid; then the use case returns to Step 2
Extension Points:	*none*
Preconditions:	The user is registered, the user has successfully queried the graph (i.e. has a graph to interact with)
Postconditions:	A graph copy is now available for the other user to access.

Use Case Name:	Load Graph
Summary:	The user can opt to load a graph from the database. This use case describes the user experience in loading a graph.
Basic Flow:	1. The use case begins when the user selects a *Load Graph* button at the top of the page. 2. The system redirects the user to a list of previously saved or shared graphs. 3. The user selects one of the graphs. 4. The system retrieves the graph, redirects the user to the graphing utility page, and displays the graph.
Alternative Flows:	● Step 2: the user may not have any previously saved/shared graphs, in which case the graph list will be empty.
Extension Points:	*Query Webroot Database, Interact With Graph, Save Graph, Share Graph*
Preconditions:	The user is registered
Postconditions:	A previously saved/shared graph (if available), is now displayed to the user and the user is free to modify the graph.