Keikaku 企画

PMR Insight Collective Knowledge (PICK)
System Design Document

Version 1.0

3/9/2020

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

Initial Release:	1.0
Current Release:	1.0
Indicator of Last Page in Document:	[END]
Date of Last Review:	03/09/2020
Date of Next Review:	03/09/2020
Target Date for Next Update:	03/09/2020

Distribution List

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Guidance Team Members:
Dr. Ann Gates, Dr. Steve Roach, Mr. Jake Lasley

Clients:
Mr. Vincent Fonseca, Mr. Baltazar Santaella, Ms. Herandy Vazquez, and Mr. Erick De Nava

Software Team Members:
Mr. Anthony Desarmier, Mr. Angel Villalpando, Mr. Mario Delgado, Mr. David Rayner,
Mr. Valentin Becerra, Mr. Jorge Garcia

Change Summary

The following table details changes made between versions of this document

Version	Date	Modifier	Description
0.1	02/26/2020	Jorge Garcia	Added Template
1.0	03/08/2020	Anthony DesArmier	Wrote class descriptions, contracts, responsibilities; wrote protocols for Event, wrote sections 2, 2.2, 2.3
1.0	03/08/2020	Jorge Garcia	Added class descriptions from CRC, wrote section 1.5 and updated references.
1.0	03/09/2020	David Rayner	Completed section 1.1 Completed Component Diagram section 2.1

Table of Contents

1. Introduction

1.1. Purpose and Intended Audience

The purpose of the Software Design Document to guide the software team in the construction of the PICK system. It is a means of construction such that the client's requirements are to be satisfied. The design or architectural overview of the system is to model the components that interact within the system. The Software Design Document is also intended for the clients to further establish or maintain the system in the future. Thus, the document must be detailed enough to provide both the software team and clients with enough information regarding the system design as possible.

1.2. Scope of Product

The PMR Insight Collective Knowledge (PICK) is the software system for which this SDD is written for. PICK is a software system to help Prevent, Mitigate, and Recover Analysts analyze vast amounts of data collected during an Adversarial Assessment (AA) by allowing them to quickly search through, view, correlate, and build visual documents which help explain the AA itself to uninvolved personnel. The customers - in this case PMR Analysts - currently must sift through the vast amounts of generated data from the AA by hand which severely hinders their workflow and efficiency in developing a report with visual aids for which to explain the nature of the AA to other personnel.

PICK will allow the customers to insert all the data generated from an AA into its system and display an organized, searchable database of that information. The customers can then quickly and efficiently find and correlate relevant data events together and help craft timelines which describe the significant events and their relations to one another during the AA. PICK will then assist the customers in crafting a visual representation of these series of events as attack graphs in order to help visualize the timeline of the AA. This assistance of analyzing the data generated by the AA and constructing visual representations of significant events will substantially reduce the time and work hours needed by the customers to understand and construct a report on the results of the AA to deliver to other personnel.

1.3. Definitions, Acronyms, and Abbreviations

1.3.1. Definitions

Table I.
Definitions of Complex Terms Found Within this Document

TERM	**DEFINITION**
Adversarial Assessment	An *Adversarial Assessment* gauges the ability of a system to support its mission(s) while withstanding validated and representative cyber threat activity [1]-[4].
Analyst	An Analyst is the end user who is evaluating log entries and associating them to their respective vectors. Their sole purpose is to generate a picture of the events taken place during an AA [1]-[4].
Log	A log is a series of noted occurrences or events that describe the events that have taken place during an AA [1]-[4].
Vector	A vector is a series of events an adversary attempts to execute in order to achieve an objective or series of objectives. In this context, Red team attack initiatives and Blue team defenses [4].
Vector Table	A vector table is a representative tabular view of vector properties this includes, vector name, vector description, and the vector's nodes [4].
Vector Element	A vector element is either a Node or a Relationship that can be added to a Vector [4].
Validator	A Validator performs the validation of the raw log files based on the criteria provided by the PMR Analyst [4].
Graphical View	A vector visualization or graphical view of the respective vector. This includes the circular nodes with image icons and their connections denoted with directional lines [4].
Event	An event is a log entry consisting of a timestamp and a textual description or message of what took place [2], [3].
Node	A node is representative of a significant event, that is an event an analyst assigns to a vector and is marked as visible on a Vector table [1]-[4].
Relationship	A relationship is a connection between a parent node to their child nodes. It is how the nodes/events are associated [1]-[4].
Attack Graph	An attack graph is a succinct representation of data structures that model all possible avenues of attacking a network [1]-[4].
White Team	White team represents the third party LSH members or analysts that observe the AA events [1]-[3].
Red Team	Red team represents the team that perform cyber-attacks on a combat system [1]-[4].
Blue Team	Blue team represents the defense team, or the team that responds to the Red team cyber-attacks [1]-[4].
Constraint	A system limitation, or the bounds to which the system can operate.
Configuration	Parameters that are defined by the Analysts regarding the AA's details [4].
Regular Expression	A search pattern consisting of a sequence of characters [1]-[3].
Property	An attribute or characteristic that belongs to an object [4].
Ingest	The intake of files [1]-[3].
Parse	The separation of text to smaller more manageable parts.
Interactive	User-system interaction, system responses to user inputs.
Prompt	Pop-up window used to display an alert or dialog of information

Enforcement Action Report	Report that outlines various errors found in validation and solicits the user for an enforcement action.
Push	The sending of committed changes to a remote repository.
Pull	The fetching and downloading of content from a remote repository and immediately updating the local repository to match that content.

1.3.2. Acronyms

Table II.
Definitions of Acronyms Found Within this Document

TERM	DEFINITION
SRS	Software Requirements Specification
PMR	Prevent Mitigate Recover [1]-[4]
PICK	PMR Insight Collective Knowledge [1]-[4]
AA	Adversarial Assessment [1]-[4]
EAR	Enforcement Action Report

1.3.3 Abbreviations

Table III.
Definitions of Abbreviations Found Within this Document

TERM	DEFINITION
e.g.	For example
i.e.	In other words
etc.	Et cetera “and” “the rest”

1.4. References

[1] V. Becerra, A. DesArmier, J. Garcia, D. Rayner and A. Villalpando, "PMR Insight Collective Knowledge (PICK) Interview Report," El Paso, 2019.

[2] Lethality, Survivability and HSI Directorate (LSH), "Software Engineering I - Fall 2019 PMR Insight Collective Knowledge (PICK)," Ramirez, Tai Elsa, El Paso, 2019.

[3] H. Vasquez , F. Larsen, F. Vicent , B. Santaella and E. De Nava, "U.S. ARMY Combat capabilities development command - Data and Analysis Center PMR Insight Collective Knowledge (PICK)," Ramirez, Tai Elsa, El Paso, 2019.

[4] Lethality, Surivability and HSI Directorate , "PICK Needs 11-6 - Updated," Ramirez, Tai Elsa, El Paso, 2019.

[5] V. Becerra, A. DesArmier, J. Garcia, D. Rayner, A. Villalpando and Mario Delgado, “Class-responsibility-collaboration Model,” El Paso, 2020.

1.5. Overview

The SDD describes the system at an architectural level providing a high-level description of subsystems, data management and other system components. Such descriptions have been provided in the following sections.

Section 2: Decomposition Description includes a System Collaboration Diagram in the form of a UML Component Diagram. This provides the system’s relationships with its subsystems. This also includes a list of all subsystem descriptions and the contracts their components fulfill.

Section 3 provides a detailed description of the User Interface subsystem. This subsystem is responsible for accepting and handling all user input. Additionally, it is also responsible for displaying all the internal system data.

Section 4 provides a detailed description of the Adversarial Assessment Data Processing subsystem. This subsystem is responsible for ingesting user provided files, relaying the files to Splunk and retrieving the data generated from said files.

Section 5 provides detailed information on the Internal Data Store subsystem. This subsystem is responsible for all the data persistence it is also responsible for all the maintenance of the configurable items in the system.

Section 6 provides detailed information on the Graph subsystem. This subsystem is responsible for the displaying and the manipulation of the node and relationship data found within a vector.

2. Decomposition Description

The decomposition description can be used by designers and maintainers to identify the major design entities of the system for purposes such as determining which entity is responsible for performing specific functions and tracing requirements to design entities. Design entities can be grouped into major classes to assist in locating information and to assist in reviewing the decomposition for completeness.

The information in the decomposition description can be used by project management for planning, monitoring, and control of a software project. They can identify each software component, its purpose, and basic functionality. This design information together with other project information can be used in estimating cost, staff, and schedule for the development effort.

2.1. System Collaboration Diagram

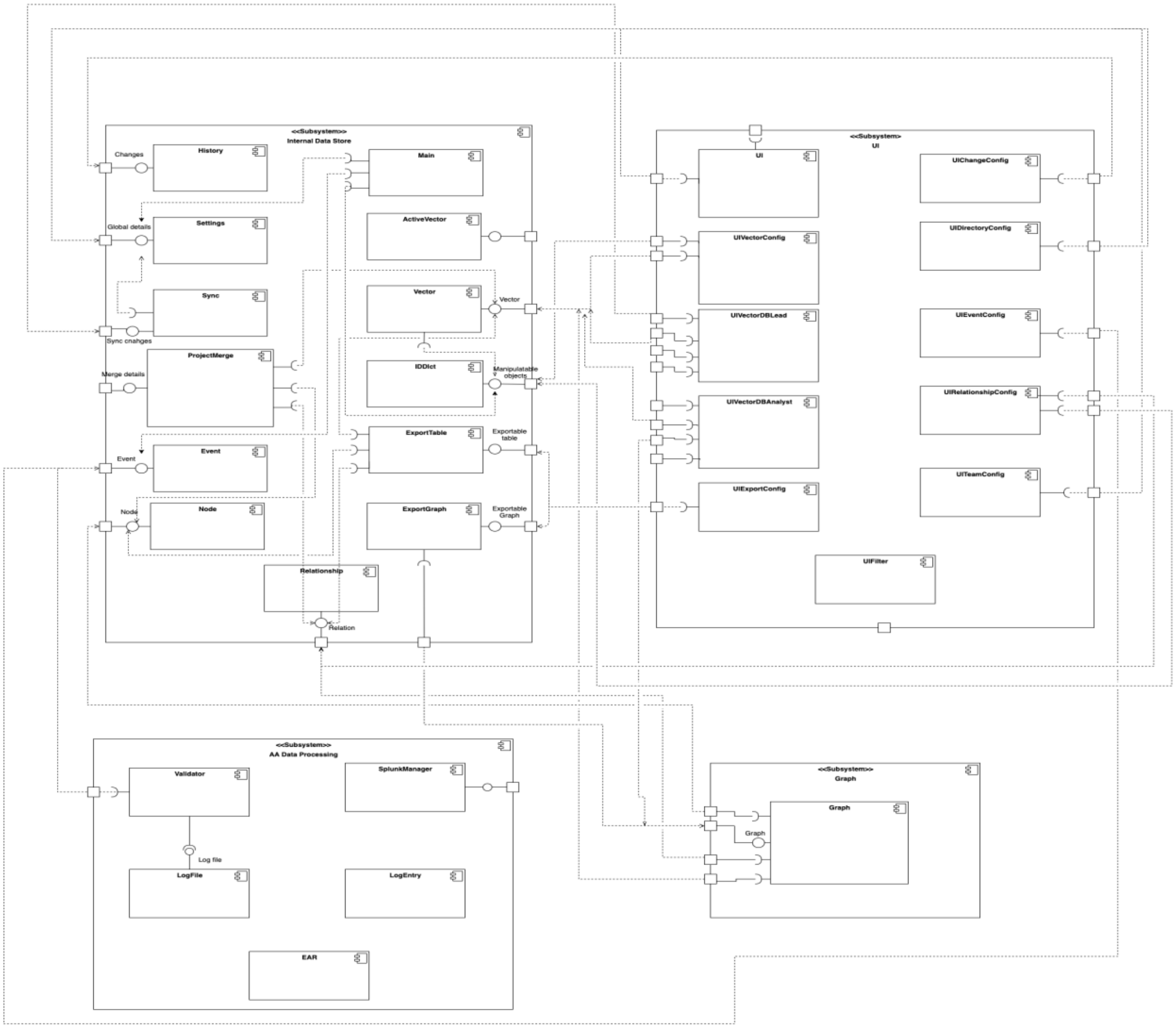

Fig. 1. UML Component Diagram for the PICK System, 2020.

The system has been structured loosely on the model-view-controller design pattern with further separation within the model component to separate distinct and independent functionality and purpose. The User Interface subsystem serves as both the view and controller components for the other subsystems. The Adversarial Assessment Data Processing, Internal Data Store, and Graph subsystems each represent a distinct and separate core function and service to the system.

2.2. Subsystem and Component Descriptions

The User Interface subsystem accepts user input for control of the other subsystems and displays internal system data. This subsystem is comprised of the following components:

1. UI: The main window which provides the bulk of the system's interface. (3.2)
    1. No Contracts
2. UiDirectoryConfig: The directory window which handles the setting of file directory paths. (3.3)
    1. No Contracts
3. UiEventConfig: The event window which handles the setting of project event information. (3.4)
    1. No Contracts
4. UiExportConfig: The export window which handles exporting vector and entry data to an external file. (3.5)
    1. No Contracts
5. UiFilterConfig: The filter window which handles the creation and application of a search filter. (3.6)
    1. Provide filter.
6. UiTeamConfig: The team window which handles the system role and connections to the host when running as a client. (3.7)
    1. No Contracts
7. UiVectorConfig: The vector window which handles the adding, editing, and deleting of vectors and their descriptions. (3.8)
    1. No Contracts
8. UiRelationshipConfig: The relationship window which handles the relationship table for the active vector. (3.9)
    1. No Contracts
9. UiVectorDBAnalyst: The analyst vector DB window which handles the pulling and pushing of vector changes between the analyst and host systems. (3.10)
    1. No Contracts
10. UiVectorDBLead: The lead vector DB window which handles the accepting of queued analyst push requests. (3.11)
    1. No Contracts
11. UiChangeConfig: The commit window which handles the committing of log entry and vector changes. (3.12)
    1. No Contracts

The Adversarial Assessment Data Processing subsystem is responsible for ingesting user-provided files, validating them, sending them to Splunk for processing, and retrieving the Splunk-generated log entries for analyst processing and manipulation in the PICK system. This subsystem is comprised of the following components:

1. LogFile: responsible for reading the user-provided directory path and importing the contents of the file. (4.2)
    1. Provide cleansing status.
    2. Provide validation status.

   3. Provide ingestion status.
   4. Save cleansing status.
   5. Save validation status.
   6. Save ingestion status.
   7. Provide log file name.
   8. Provide enforcement action report.
   9. Save enforcement action report.
2. LogEntry: responsible for storing the indexed entries in a log file. (4.3)
   1. Provide entry timestamp.
   2. Provide entry description.
   3. Provide log file name.
3. Validator: responsible for validating and cleansing the content of user-provided files. (4.4)
   1. Cleanse and validate log files.
4. EnforcementActionReport: responsible for a report that outlines various errors found in validation. (4.5)
   1. Provide enforcement action report
5. SplunkManager: responsible for the interface between Splunk and the system to submit log files and retrieve log entries. (4.6)
   1. Provide log files to Splunk.
   2. Provide parsed log entries.

The Internal Data Store subsystem is responsible for the maintenance and persistence of user-provided, configurable items. Such items can include customized event names, vector names, vector components such as nodes and relationships, and any other labels or data that the system allows the user to personalize. This subsystem is comprised of the following components:

1. Main: serves as an entry and exit point to the system. (5.2)
   1. No Contracts
2. Event: responsible for maintaining the information and attributes that describe the specific event for an analysis/session. (5.3)
   1. Provide event name.
   2. Provide event description.
   3. Provide start timestamp.
   4. Provide event end timestamp.
   5. Save event name.
   6. Save event description.
   7. Save start timestamp.
   8. Save event end timestamp.
   9. Save event object.
   10. Load event object.
3. Settings: responsible for housing global variables of a given event/session. (5.4)
   1. Provide lead status check.
   2. Provide host machine IP address.
   3. Provide lead machine IP address.
   4. Provide red team directory.
   5. Provide blue team directory.
   6. Provide white team directory.
   7. Provide root directory.
   8. Provide icon directory.
   9. Save lead status check.
   10. Save host machine IP address.
   11. Save lead machine IP address.

12. Save red team directory.
13. Save blue team directory.
14. Save white team directory.
15. Save root directory.
16. Save icon directory.
17. Save settings object.
18. Load settings object.

4. IDDict: responsible for storing a map of generic objects and their unique id. (5.5)
    1. Add an entry.
    2. Remove an entry.
    3. Retrieve an entry.
    4. Retrieve all entries.
    5. Save IDDict.
    6. Load IDDict.
5. Vector: responsible for storing the data of nodes and relationships that compose a vector. (5.6)
    1. Provide vector name.
    2. Provide vector description.
    3. Save vector name.
    4. Save vector description.
    5. Add a node.
    6. Remove a node.
    7. Retrieve a node.
    8. Retrieve all nodes.
    9. Add a relationship.
    10. Remove a relationship.
    11. Retrieve a relationship.
    12. Retrieve all relationships.
    13. Provide node property visibilities.
    14. Save node property visibilities.
    15. Save settings object.
    16. Load settings object.
6. ActiveVector: responsible for storing the currently selected vector and its unique id. (5.7)
    1. Provide active vector.
    2. Provide active vector id.
    3. Save active vector.
    4. Save active vector id.
7. Node: responsible for the storage of log entry properties. (5.8)
    1. Provide node name.
    2. Provide node timestamp.
    3. Provide node description.
    4. Provide node log entry reference.
    5. Provide node log creator.
    6. Provide node event type.
    7. Provide node icon type.
    8. Provide node source file.
    9. Provide node visibility.
    10. Provide node coordinate position.
    11. Save node name.
    12. Save node timestamp.
    13. Save node description.

14. Save node log entry reference.
15. Save node log creator.
16. Save node event type.
17. Save node icon type.
18. Save node source filename.
19. Save node visibility.
20. Save node coordinate position.

8. Relationship: responsible for storing the child and parent node IDs that constitute a given relationship in the system. (5.9)
   1. Provide parent node identification number.
   2. Provide child node identification number.
   3. Provide relationship label.
   4. Provide relationship line coordinates.
   5. Save parent node identification number.
   6. Save child node identification number.
   7. Save relationship label.
   8. Save relationship line coordinates.
9. History: responsible for storing the vectors', nodes' and relationships' history of changes and saving them. (5.10)
   1. Display history of changes.
   2. Save changes.
10. ProjectMerge: responsible for the merging of analyst and lead history changes.
    1. Merge data. (5.11)
11. Sync: responsible for handling data transferal from lead and client. (5.12)
    1. Provides push request changes to lead.
    2. Provide pull request changes to client.
    3. Provides push request queue.
    4. Remove push request from queue.
12. ExportTable: responsible for the formatting of the log entry tables that compose the vectors of a given event into an exportable file. (5.13)
    1. Save tables to file.
13. ExportGraph: responsible for the formatting of the final graph with its corresponding vector into an exportable image file. (5.14)
    1. Save graph to image file.

The Graph subsystem is responsible for the displaying and the manipulation of the node and relationship data found within a vector. This subsystem is comprised of the following component:

1. Graph: responsible for rendering nodes, relationships, their relative positions, and their properties for a given vector. (6.2)
   1. Provide graph orientation.
   2. Provide timeline interval.
   3. Save graph orientation.
   4. Save timeline interval.
   5. Render nodes.
   6. Render relationships.
   7. Move node.
   8. Move relationship.

2.3. Dependencies

Since the User Interface subsystem displays and controls all other subsystems, this subsystem needs to be developed first. The Adversarial Assessment Data Processing, Internal Data Store, and Graph subsystem are patterned on the specific workflow sequence of the system and thus each subsystem is generally dependent on the previous subsystem to perform any meaningful functions. Therefore, the Adversarial Assessment Data Processing, Internal Data Store, and Graph subsystem should be developed in sequence.

3. User Interface: Detailed Description

3.1. Component Description

The User Interface subsystem accepts user input for control of the other subsystems and displays internal system data. This subsystem is composed of the Ui, UiDirectoryConfig, UiEventConfig, UiExportConfig, UiFilterConfig, UiTeamConfig, UiVectorConfig, UiRelationshipConfig, UiVectorDBAnalyst, UiVectorDBLead, and the UiChangeConfig classes.

3.2. Class Description: Ui

The main window which provides the bulk of the system's interface.

Class Name: Ui	
Superclass: QMainWindow	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Launches UiChangeConfig.	
• Launches UiDirectoryConfig.	
• Launches UiEventConfig.	
• Launches UiExportConfig.	
• Launches UiFilterConfig.	
• Launches UiTeamConfig.	
• Launches UiVectorConfig.	
• Launches UiRelatonshipConfig.	
• Launches UiVectorDBAnalyst when not lead.	Settings 1
• Launches UiVectorDBLead when lead.	Settings 1
• Knows the vector IDDict.	
• Knows the log file IDDict.	
• Knows the log entry IDDict.	
• Discovers log files.	Settings 4-7; IDDict 1
• Displays log file table.	IDDict 3; LogFile 1-3, 7, 9
• Displays Enforcement Action Report table.	EnforcementActionReport 1
• Selects an Enforcement Action Report to display.	LogFile 9
• Cleanses log files.	Validator 1-2
• Validates log files.	Validator 3
• Ingests log files.	SplunkManager 1
• Retrieves indexed log entries.	SplunkManager 2; IDDict 1
• Displays log entry table.	IDDict 4; LogEntry 1-3
• Assigns log entries to vectors.	Vector 1-2
• Sets active vector.	IDDict 3; ActiveVector 2-4
• Displays active vector name.	ActiveVector 1; Vector 1
• Displays active vector description.	ActiveVector 1; Vector 2
• Displays node table.	ActiveVector 1; Vector 1-2
• Adds nodes.	ActiveVector 1; Vector 5
• Edits node properties.	ActiveVector 1; Vector 7; 11-19
• Knows node visibility.	ActiveVector 1; Vector 7; Node 9
• Sets node visibility.	ActiveVector 1; Vector 7; Node 19

• Deletes nodes.	ActiveVector 1; Vector 6
• Displays node property visibilities.	ActiveVector 1; Vector 13
• Selects node property visibilities.	ActiveVector 1; Vector 14
• Displays vector graphical view.	ActiveVector 1; Graph 5
• Zooms in/out on the graphical view.	
• Knows the timeline orientation.	Graph 1
• Knows the timeline interval.	Graph 2
• Sets the timeline orientation.	Graph 3
• Sets the timeline interval.	Graph 4
No Contracts	

3.3. Class Description: UiDirectoryConfig

The directory window which handles the setting of file directory paths.

Class Name: UiDirectoryConfig	
Superclass: QDialog	
Subclasses: None	
Private Responsibilities	**Collaborations**
• Displays user provided directory path for root ingestion.	Settings 7
• Displays red team directory path.	Settings 4
• Displays blue team directory path.	Settings 5
• Displays white team directory path.	Settings 6
• Collects user provided directory path for root ingestion.	Settings 15
• Collects red team directory path.	Settings 12
• Collects blue team directory path.	Settings 13
• Collects white team directory path.	Settings 14
No Contracts	

3.4. Class Description: UiEventConfig

The event window which handles the setting of project event information.

Class Name: UiEventConfig	
Superclass: QDialog	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Displays event name.	Event 1
• Displays event description.	Event 2
• Displays event start timestamp.	Event 3
• Displays event end timestamp.	Event 4
• Collects event name.	Event 5
• Collects event description.	Event 6
• Collects event start timestamp.	Event 7
• Collects event end timestamp.	Event 8
• Saves event information to file.	Event 9
No Contracts	

3.5. Class Description: UiExportConfig

The export window which handles exporting vector and entry data to an external file.

Class Name: UiExportConfig	
Superclass: QDialog	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Collects specified directory. • Collects directory file format. • Collects name for exported file. • Creates an image file of vector graph. • Creates a csv file of vector table.	ExportGraph 1 ExportTable 1
No Contracts	

3.6. Class Description: UiFilterConfig

The filter window which handles the creation and application of a search filter.

Class Name: UiFilterConfig	
Superclass: QDialog	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Collects keywords to search for in generated log entries, nodes, and relationships. • Collects the start/end timestamp search-criteria to apply to generated log entries and nodes. • Collects the “creator” (red, white, or blue) search criteria for generated log entries and nodes. • Collects the “event type” (red, white, or blue) search-criteria for generated log entries and nodes.	
Contract: 1. Provide filter.	
Responsibilities	**Collaborations**
1. Knows keywords. 2. Knows timestamps. 3. Knows “creator”. 4. Knows “event type”.	

3.7. Class Description: UiTeamConfig

The team window which handles the system role and connections to the host when running as a client.

Class Name: UiTeamConfig	
Superclass: QDialog	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Displays lead status. • Displays host machine IP address. • Displays lead machine IP address. • Collects lead status. • Collects host machine IP address. • Collects lead machine IP address.	Settings 1 Settings 2 Settings 3 Settings 9 Settings 10 Settings 11

- Displays number of connections to the lead machine IP address.	
No Contracts	

3.8. Class Description: UiVectorConfig

The vector window which handles the adding, editing, and deleting of vectors and their descriptions.

Class Name: UiVectorConfig	
Superclass: QFrame	
Subclasses: None	
Private Responsibilities:	**Collaborations**
- Displays vector name. - Displays vector description. - Collects vector name. - Collects vector description. - Adds a new vector. - Deletes a vector. - Edits existing vector attributes.	IDDict 4; Vector 1 IDDict 4; Vector 2 IDDict 1 IDDict 2 IDDict 3; Vector 3-4
No Contracts	

3.9. Class Description: UiRelationshipConfig

The relationship window which handles the relationship table for the active vector.

Class Name: UiRelationshipConfig	
Superclass: QFrame	
Subclasses: None	
Private Responsibilities:	**Collaborations**
- Display relationship id.	IDDict 4
- Display parent id.	IDDict 4; Relationship 1
- Display child id.	IDDict 4; Relationship 2
- Display label.	IDDict 4; Relationship 3
- Collects parent id.	
- Collects child id.	
- Collects label description.	
- Adds a new relationship.	IDDict 1
- Deletes a relationship.	IDDict 2
- Edits existing relationship attributes.	IDDict 3; Relationship 5-7
No Contracts	

3.10. Class Description: UiVectorDBAnalyst

The analyst vector DB window which handles the pulling and pushing of vector changes between the analyst and host systems.

Class Name: UiVectorDBAnalyst	
Superclass: QFrame	
Subclasses: None	
Private Responsibilities:	**Collaborations**
- Displays lead’s pull table.	Sync 2; IDDict 3; Vector 1-2; Graph 5

• Displays client’s push table.	Vector 1-2; Graph 5
• Initiates pull request from lead’s table.	Sync 2; IDDict 3; Vector 1-2
• Initiates push request from client’s table.	Sync 1; IDDict 3; Vector 1-2
No Contracts	

3.11. Class Description: UiVectorDBLead

The lead vector DB window which handles the accepting of queued analyst push requests.

Class Name: UiVectorDBLead	
Superclass: QFrame	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Knows pending push requests from clients.	Sync 3
• Accept push requests to the database.	ProjectMerge 2, Sync 4
• Declines push requests to the database.	Sync 4
• Display information to be pushed to the database.	Vector 1-2; Graph 5
No Contracts	

3.12. Class Description: UiChangeConfig

The commit window which handles the committing of log entry and vector changes.

Class Name: UiChangeConfig	
Superclass: QDialog	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Display vector changes.	History 1
• Display node changes.	History 1
• Display relationship changes.	History 1
• Saves vector, node, and relationship changes.	History 2
No Contracts	

4. Adversarial Assessment Data Processing: Detailed Description

4.1. Component Description

The Adversarial Assessment Data Processing subsystem is responsible for ingesting user-provided files, validating them, sending them to Splunk for processing, and retrieving the Splunk-generated log entries for analyst processing and manipulation in the PICK system. This subsystem includes the LogFile, LogEntry, Validator, EnforcementActionReport, and SplunkManager classes.

4.2. Class Description: LogFile

The LogFile class is responsible for reading the user-provided directory path and importing the contents of the file.

Class Name: LogFile	
Superclass: None	
Subclasses: None	
Private Responsibilities: - Creates copy of cleansed file.	
Contract: 1. Provide cleansing status.	
Responsibilities	**Collaborations**
1. Knows cleansing status.	
Contract: 2. Provide validation status.	
Responsibilities	**Collaborations**
2. Knows validation status.	
Contract: 3. Provide ingestion status.	
Responsibilities	**Collaborations**
3. Knows ingestion status.	
Contract: 4. Save cleansing status.	
Responsibilities	**Collaborations**
4. Records cleaning status.	
Contract: 5. Save validation status.	
Responsibilities	**Collaborations**
5. Records validation status.	
Contract: 6. Save ingestion status.	
Responsibilities	**Collaborations**
6. Records ingestion status.	
Contract: 7. Provide log file name.	
Responsibilities	**Collaborations**
7. Knows log file name.	
Contract: 8. Provide enforcement action report.	
Responsibilities	**Collaborations**
8. Knows enforcement action report.	
Contract: 9. Save enforcement action report.	
Responsibilities	**Collaborations**
9. Records enforcement action report.	

4.3. Class Description: LogEntry

The LogEntry class is responsible for storing the indexed entries in a log file.

Class Name: LogEntry	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide entry timestamp.	
Responsibilities	**Collaborations**
1. Knows entry timestamp.	
Contract: 2. Provide entry description.	
Responsibilities	**Collaborations**
2. Knows entry description.	
Contract: 3. Provide log file name.	
Responsibilities	**Collaborations**
3. Knows log file name	

4.4. Class Description: Validator

The Validator class is responsible for validating and cleansing the content of user-provided files.

Class Name: Validator	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Cleanse and validate log files.	
Responsibilities	**Collaborations**
1. Cleanses TMUX log files for unwanted characters.	
2. Cleanses log files of empty lines/rows.	
3. Validates log files' timestamps.	Event 3-4
4. Reports cleansing status result.	LogFile 4
5. Reports validation status result.	LogFile 5
6. Generates an enforcement action report.	LogFile 9

4.5. Class Description: EnforcementActionReport

The EnforcementActonReport is responsible for a report that outlines various errors found in validation.

Class Name: EnforcementActionReport	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide enforcement action report.	
Responsibilities	**Collaborations**
1. Knows errors found within log file.	

4.6. Class Description: SplunkManager

The SplunkManager class is responsible for the interface between Splunk and the system to submit log files and retrieve log entries.

Class Name: SplunkManager
Superclass: None

Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide log files to Splunk.	
Responsibilities	**Collaborations**
1. Sends log files to Splunk.	
Contract: 2. Provide parsed log entries.	
Responsibilities	**Collaborations**
2. Retrieves parsed log entries.	

5. Internal Data Store: Detailed Description

5.1. Component Description

The Internal Data Store subsystem is responsible for the maintenance and persistence of user-provided, configurable items. Such items can include customized event names, vector names, and any other labels or data that the system allows the user to personalize. This subsystem includes the Main, Event, Settings, Dictionary, QObject, Vector, ActiveVector, Node, Relationship, History, ProjectMerge, ExportTable, and ExportGraph classes.

5.2. Class Description: Main

Serves as an entry and exit point to the system.

Class Name: Main	
Superclass: None	
Subclasses: None	
Private Responsibilities:	**Collaborations**
• Loads the event object from file. • Loads the settings object from file. • Loads the vector IDDict from file. • Loads the log file IDDict from file. • Loads the log entry IDDict from file. • Saves the log file IDDict to file. • Saves the log entry IDDict to file. • Saves the settings object to file. • Launches UI.	Event 10 Settings 18 IDDict 6 IDDict 6 IDDict 6 IDDict 5 IDDict 5 Settings 17
No Contracts	

5.3. Class Description: Event

The Event class is responsible for maintaining the information and attributes that describe the specific event for an analysis/session.

Class Name: Event	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide event name.	
Responsibilities	**Collaborations**
1. Knows user-provided event name.	
def get_name() -> str: returns: String representing the event name. Retrieves the name from this event. Pre: None Post: Returned string is non-null.	
Contract: 2. Provide event description.	
Responsibilities	**Collaborations**
2. Know user-provided event description.	
def get_desc() -> str: returns: String representing the event description.	

Retrieves the description from this event. Pre: None Post: Returned string is non-null.	
Contract: 3. Provide start timestamp.	
Responsibilities	**Collaborations**
3. Knows user-provided start timestamp.	
def get_start_time() -> QDateTime: returns: QDateTime representing the start timestamp. Retrieves the start timestamp from this event. Pre: None Post: Returned QDateTime is non-null.	
Contract: 4. Provide event end timestamp.	
Responsibilities	**Collaborations**
4. Knows user-provided end timestamp.	
def get_end_time() -> QDateTime: returns: QDateTime representing the end timestamp. Retrieves the end timestamp from this event. Pre: None Post: Returned QDateTime is non-null.	
Contract: 5. Save event name.	
Responsibilities	**Collaborations**
5. Records user-provided event name.	
def set_name(name: str): name: String representing the event name. Sets the name of this event. Pre: name is non-null. Post: name is set to settings attribute name.	
Contract: 6. Save event description.	
Responsibilities	**Collaborations**
6. Records user-provided event description.	
def set_desc(desc: str): desc: String representing the event description. Sets the description of this event. Pre: desc is non-null. Post: desc is set to settings attribute desc.	
Contract: 7. Save start timestamp.	
Responsibilities	**Collaborations**
7. Records user-provided start timestamp.	
def set_start_time(start: QDateTime): start: QDateTime representing the start timestamp. Sets the start timestamp of this event. Pre: start is non-null. Post: If start is set before attribute end_timestamp, start is set to attribute start_timestamp; otherwise, start_timestamp is set to end_timestamp.	
Contract: 8. Save event end timestamp.	
Responsibilities	**Collaborations**
8. Records user-provided end timestamp.	
def set_end_time(): start: QDateTime representing the end timestamp.	

Sets the end timestamp of this event. Pre: end is non-null. Post: If end is set after attribute start_timestamp, end is set to attribute end_timestamp; otherwise, end_timestamp is set to start_timestamp.	
Contract: 9. Save event object.	
Responsibilities	**Collaborations**
9. Saves itself to file.	
def save(): Saves this settings object to a file. Pre: None Post: Settings object binary is written to a file in ./cache/settings.pk; throws IOException.	
Contract: 10. Load event object.	
Responsibilities	**Collaborations**
10. Loads itself from file.	
def load(): Loads a settings object from a file. Pre: None Post: If file './cache/settings.pk' exists, settings object is initialized from the file; if file does not exist, nothing happens; throws IOException.	

5.4. Class Description: Settings

The Settings class is responsible for housing global variables of a given event/session.

Class Name: Settings	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide lead status check.	
Responsibilities	**Collaborations**
1. Knows lead status check.	
Contract: 2. Provide host machine IP address.	
Responsibilities	**Collaborations**
2. Knows host machine IP address.	
Contract: 3. Provide lead machine IP address.	
Responsibilities	**Collaborations**
3. Knows lead machine IP address.	
Contract: 4. Provide red team directory.	
Responsibilities	**Collaborations**
4. Knows red team directory.	
Contract: 5. Provide blue team directory.	
Responsibilities	**Collaborations**
5. Knows blue team directory.	
Contract: 6. Provide white team directory.	
Responsibilities	**Collaborations**
6. Knows white team directory.	
Contract: 7. Provide root directory.	
Responsibilities	**Collaborations**
7. Knows root directory.	
Contract: 8. Provide icon directory.	
Responsibilities	**Collaborations**

8. Knows icon directory.	
Contract: 9. Save lead status check.	
Responsibilities	**Collaborations**
9. Records lead status check.	
Contract: 10. Save host machine IP address.	
Responsibilities	**Collaborations**
10. Records host machine IP address.	
Contract: 11. Save lead machine IP address.	
Responsibilities	**Collaborations**
11. Records lead machine IP address.	
Contract: 12. Save red team directory.	
Responsibilities	**Collaborations**
12. Records red team directory.	
Contract: 13. Save blue team directory.	
Responsibilities	**Collaborations**
13. Records blue team directory.	
Contract: 14. Save white team directory.	
Responsibilities	**Collaborations**
14. Records white team directory.	
Contract: 15. Save root directory.	
Responsibilities	**Collaborations**
15. Records root directory.	
Contract: 16. Save icon directory.	
Responsibilities	**Collaborations**
16. Records icon directory.	
Contract: 17. Save settings object.	
Responsibilities	**Collaborations**
17. Saves itself to file.	
Contract: 18. Load settings object.	
Responsibilities	**Collaborations**
18. Loads itself from file.	

5.5. Class Description: IDDict

The IDDict class is responsible for storing a map of generic objects and their unique id.

Class Name: IDDict	
Superclass: QObject	
Subclasses: None	
Private Responsibilities: • Maps a generated uid for each entry.	
Contract: 1. Add an entry.	
Responsibilities	**Collaborations**
1. Adds an entry to the dictionary.	
Contract: 2. Remove an entry.	
Responsibilities	**Collaborations**
2. Removes an entry from the dictionary.	
Contract: 3. Retrieve an entry.	
Responsibilities	**Collaborations**
3. Retrieves an entry from the dictionary.	

Contract: 4. Retrieve all entries.	
Responsibilities	**Collaborations**
4. Retrieves all entries from the dictionary.	
Contract: 4. Save IDDict.	
Responsibilities	**Collaborations**
5. Saves IDDict to file.	
Contract: 5. Load IDDict.	
Responsibilities	**Collaborations**
6. Loads IDDict from file.	

5.6. Class Description: Vector

The vector class is responsible for storing the data of nodes and relationships that compose a vector.

Class Name: Vector	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide vector name.	
Responsibilities	**Collaborations**
1. Knows vector name.	
Contract: 2. Provide vector description.	
Responsibilities	**Collaborations**
2. Knows vector description.	
Contract: 3. Save vector name.	
Responsibilities	**Collaborations**
3. Records vector name.	
Contract: 4. Save vector description.	
Responsibilities	**Collaborations**
4. Records vector description.	
Contract: 5. Add a node.	
Responsibilities	**Collaborations**
5. Add node to dictionary.	IDDict 1
Contract: 6. Remove a node.	
Responsibilities	**Collaborations**
6. Remove node from dictionary.	IDDict 2
Contract: 7. Retrieve a node.	
Responsibilities	**Collaborations**
7. Retrieve node form dictionary.	IDDict 3
Contract: 8. Retrieve all nodes.	
Responsibilities	**Collaborations**
8. Retrieve all nodes form dictionary.	IDDict 4
Contract: 9. Add a relationship.	
Responsibilities	**Collaborations**
9. Add relationship to dictionary.	IDDict 1
Contract: 10. Remove a relationship.	
Responsibilities	**Collaborations**
10. Remove relationship from dictionary.	IDDict 2
Contract: 11. Retrieve a relationship.	

Responsibilities	**Collaborations**
11. Retrieve relationship form dictionary.	IDDict 3
Contract: 12. Retrieve all relationships.	
Responsibilities	**Collaborations**
12. Retrieve all relationships form dictionary.	IDDict 4
Contract: 13. Provide node property visibilities.	
Responsibilities	**Collaborations**
13. Knows node property visibility flags.	
Contract: 14. Save node property visibilities.	
Responsibilities	**Collaborations**
14. Records node property visibility flags.	
Contract: 15. Save settings object.	
Responsibilities	**Collaborations**
15. Saves itself to file.	
Contract: 16. Load settings object.	
Responsibilities	**Collaborations**
16. Loads itself from file.	

5.7. Class Description: ActiveVector

The ActiveVector class is responsible for storing the currently selected vector and its unique id.

Class Name: ActiveVector	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide active vector.	
Responsibilities	**Collaborations**
1. Knows active vector.	
Contract: 2. Provide active vector id.	
Responsibilities	**Collaborations**
2. Knows active vector id.	
Contract: 3. Save active vector.	
Responsibilities	**Collaborations**
3. Records active vector.	
Contract: 4. Save active vector id.	
Responsibilities	**Collaborations**
4. Records active vector id.	

5.8. Class Description: Node

The Node class is responsible for the storage of log entry properties.

Class Name: Node	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide node name.	
Responsibilities	**Collaborations**
1. Knows node name.	
Contract: 2. Provide node timestamp.	
Responsibilities	**Collaborations**

2. Knows node timestamp.	
Contract: 3. Provide node description.	
Responsibilities	**Collaborations**
3. Knows node description.	
Contract: 4. Provide node log entry reference.	
Responsibilities	**Collaborations**
4. Knows node log entry reference.	
Contract: 5. Provide node log creator.	
Responsibilities	**Collaborations**
5. Knows node log creator.	
Contract: 6. Provide node event type.	
Responsibilities	**Collaborations**
6. Knows node event type.	
Contract: 7. Provide node icon type.	
Responsibilities	**Collaborations**
7. Knows node icon type.	
Contract: 8. Provide node source file.	
Responsibilities	**Collaborations**
8. Knows node source filename.	
Contract: 9. Provide node visibility.	
Responsibilities	**Collaborations**
9. Knows node visibility.	
Contract: 10. Provide node coordinate position.	
Responsibilities	**Collaborations**
10. Knows node coordinate position.	
Contract: 11. Save node name.	
Responsibilities	**Collaborations**
11. Records node name.	
Contract: 12. Save node timestamp.	
Responsibilities	**Collaborations**
12. Records node timestamp.	
Contract: 13. Save node description.	
Responsibilities	**Collaborations**
13. Records node description.	
Contract: 14. Save node log entry reference.	
Responsibilities	**Collaborations**
14. Records node log entry reference.	
Contract: 15. Save node log creator.	
Responsibilities	**Collaborations**
15. Records node log creator.	
Contract: 16. Save node event type.	
Responsibilities	**Collaborations**
16. Records node event type.	
Contract: 17. Save node icon type.	
Responsibilities	**Collaborations**
17. Records node icon type.	
Contract: 18. Save node source filename.	
Responsibilities	**Collaborations**
18. Records node source filename.	

Contract: 19. Save node visibility.	
Responsibilities	**Collaborations**
19. Records node visibility.	
Contract: 20. Save node coordinate position.	
Responsibilities	**Collaborations**
20. Records node coordinate position.	

5.9. Class Description: Relationship

The Relationship class is responsible for storing the child and parent node IDs that constitute a given relationship in the system.

Class Name: Relationship	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide parent node identification number.	
Responsibilities	**Collaborations**
1. Knows parent node identification number.	
Contract: 2. Provide child node identification number.	
Responsibilities	**Collaborations**
2. Knows child node identification number.	
Contract: 3. Provide relationship label.	
Responsibilities	**Collaborations**
3. Knows relationship label.	
Contract: 4. Provide relationship line coordinates.	
Responsibilities	**Collaborations**
4. Knows relationship line coordinates.	
Contract: 5. Save parent node identification number.	
Responsibilities	**Collaborations**
5. Records parent node identification number.	
Contract: 6. Save child node identification number.	
Responsibilities	**Collaborations**
6. Records child node identification number.	
Contract: 7. Save relationship label.	
Responsibilities	**Collaborations**
7. Records relationship label.	
Contract: 8. Save relationship line coordinates.	
Responsibilities	**Collaborations**
8. Records relationship line coordinates.	

5.10. Class Description: History

The History class is responsible for storing the vectors', nodes' and relationships' history of changes and saving them.

Class Name: History	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Display history of changes.	
Responsibilities	**Collaborations**

1. Displays changes to vector, node, and relationship tables.	IDDict 4; Vector 1-2, 8, 12; Node 1-10; Relationship 1-4
Contract: 2. Save changes.	
Responsibilities	**Collaborations**
2. Saves changes made to vector, node, and relationship tables.	IDDict 5

5.11. Class Description: ProjectMerge

The ProjectMerge class is responsible for the merging of analyst and lead history changes.

Class Name: ProjectMerge	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Merge data.	
Responsibilities	**Collaborations**
1. Merges vector, node, and relationship data with stored vector, node, and relationship data.	Vector 1-5, 8-9, 12; Node 1-9, 11-19; Relationship 1-8

5.12. Class Description: Sync

The Sync class is responsible for handling data transferal from lead and client.

Class Name: Sync	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provides push request changes to lead.	
Responsibilities	**Collaborations**
1. Pushes client changes to lead.	Settings 3
Contract: 2. Provide pull request changes to client.	
Responsibilities	**Collaborations**
2. Pulls client changes to lead.	Settings 3
Contract: 3. Provides push request queue.	
Responsibilities	**Collaborations**
3. Queues push requests from clients.	
Contract: 4. Remove push request from queue.	
Responsibilities	**Collaborations**
4. Removes push request from queue.	

5.13. Class Description: ExportTable

The ExportTable class is responsible for the formatting of the log entry tables that compose the vectors of a given event into an exportable file.

Class Name: ExportTable	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Save tables to file.	
Responsibilities	**Collaborations**
1. Compiles the node and relationship table into a single, exportable file.	Vector 1-2, 8, 12; Node 1-9; Relationship 1-3

5.14. Class Description: ExportGraph

The ExportGraph class is responsible for the formatting of the final graph with its corresponding vector into an exportable image file.

Class Name: ExportGraph	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Save graph to image file.	
Responsibilities	**Collaborations**
1. Renders an exportable image file of the Graph.	Graph 5-6

6. Graph: Detailed Description

6.1. Component Description

The Graph subsystem, which only includes one component Graph, is responsible for the displaying and the manipulation of the node and relationship data found within a vector.

6.2. Class Description: Graph

The Graph class is responsible for rendering nodes, relationships, their relative positions, and their properties for a given vector.

Class Name: Graph	
Superclass: None	
Subclasses: None	
Private Responsibilities: None	
Contract: 1. Provide graph orientation.	
Responsibilities	**Collaborations**
1. Knows graph orientation.	
Contract: 2. Provide timeline interval.	
Responsibilities	**Collaborations**
2. Knows timeline interval.	
Contract: 3. Save graph orientation.	
Responsibilities	**Collaborations**
3. Records graph orientation.	
Contract: 4. Save timeline interval.	
Responsibilities	**Collaborations**
4. Records timeline interval.	
Contract: 5. Render nodes.	
Responsibilities	**Collaborations**
5. Renders nodes.	Vector 8; Node 1-10
Contract: 6. Render relationships.	
Responsibilities	**Collaborations**
6. Renders relationships.	Vector 12; Relationship 1-4
Contract: 7. Move node.	
Responsibilities	**Collaborations**
7. Moves node.	Vector 7; Node 20
Contract: 8. Move relationship.	
Responsibilities	**Collaborations**
8. Moves relationship.	Vector 11; Relationship 4

[END]