1 Project Overview

The CERT Executable Code Analysis team develops and maintains a set of program analysis tools, known as Pharos that are based on an increasingly complex infrastructure. There are many tools in the Pharos suite(see https://insights.sei.cmu.edu/sei_blog/2017/08/pharos-binary-static-analysis- tools-relea sed-on-github.html) and they perform a variety of functions to automate and support program analysis and reverse engineering tasks. Pharos tools typically export results in rudimentary data formats that are then ingested by other analytical tools. Despite generally using the same underlying libraries and infrastructure, the Pharos toolset is not easy to operate, inflexible, and individual tools do not integrate well together or with external tools.

2 Goal

The goal of this project is to develop a control center to unify the pharos tool set and make it easier to manage. This control center will enable program analysts to run, monitor, and export results from Pharos tools using a single interface. The ultimate vision would be to develop a project-based model where each tool can be applied on a common set of artifacts as needed, state is preserved as tools are run, and the results are easy to review and consume. Finally, we also want to support different deployment models (local versus distributed) to enable more sophisticated processing.

3 Requirements

3.1 Users

1. Analysts: They mainly use Pharos without Internet to avoid privacy leak. An isolated environment for the executable Pharos tool is what they really want. They may use offline version Pharos tool to analyze their their software in local.

2. Researchers: They are active users that don't care about connecting the Internet or upload their binaries. Those users may prefer the convenience. A central server version of Pharos tool is more suitable for them.

3.2 Customer

To maintain the functionality in Pharos, the customer (developer of Pharos) need to continuous updated the Pharos, which requires us to decouple the Pharos's kernel to our control layer and an online kernel updating might be a choice.

3.3 Technical

1. Users: The users should be able to start any Pharos tools and would able to store and check the status and output of each job. For the jobs which execute too much time, the user should be able to kill such job. All the output and status file need to be available for users to download.

2. Control Flow: The user might have ordered executions of Pharos, and server may provide Multi-task support to speed up.

3. Environment: When users try to run the Pharos job, because the executable binaries might be malicious, they may need to run it in a isolated environments like docker containers and users might want to have some global configuration on this project.

4. Usability: Provide good interaction manuscript for both Pharos and Felucca in UI.

4 Prioritized Requirements

4.1 Mandatory Requirements

1. Pharos Functionality: Providing Offline Pharos execution logic, including binary executable file and signature file management, output file storage and download, chain execution of Pharos tools and concurrency between jobs submitted by users.

2. Kernel Update: Developers may update the Pharos kernel, so the control panel should be independent from the kernel, making it easy to update locally or remotely.

4.2 Better to implement Requirements

1. Online Version of Pharos: Provide online version for researches. Since they don't care much about the security and they are happy to share their data result with others. We need to deployed a central server to handle user requests.

2. Isolation: The binary executable maybe dangerous, we need to keep it isolated with user work space, a better way to do this is to create a container(like docker) for these data.

4.3 Optional Requirements

1. Create a user-friendly UI and maintain it with every updating

2. Provide good interaction manuscript for both Pharos tools and Felucca UI

3. Load balancing optimization of online version Pharos tools

5 Obsolete Plans

At present, we have three different plans with different requirements on storage.

1. The stateful one. A database is required for storing username, password, binaries and execution records. Users can upload their binaries and then run those binaries with different parameters for multiple times. After users submit a request, they can return soon and check the output from a list of execution records.

2. The stateless one. A database is not necessary. Each time user just upload their binaries with parameters, and then they must wait until the server sends the output back.

3. Mix of above. A database is required but only for storing username, password and execution records. The server won't save users' binaries thus there is less pressure on storage. Users don't need to wait for the output since they can check the record later.

6 Plan(2.23)

1. The backend. It must support full functionality to work under offline mode. It will contain a database for data persistency. For isolation, execution will be only in the docker.

2. The frontend. It aims to show the UI interface and is for users' convenience. It can work on both offline mode and online mode. When it is used for online mode, the communication between the server and the client(browser) must be encrypted and authenticated.

7 Challenge

8 Stakeholders

9 Business Workflow

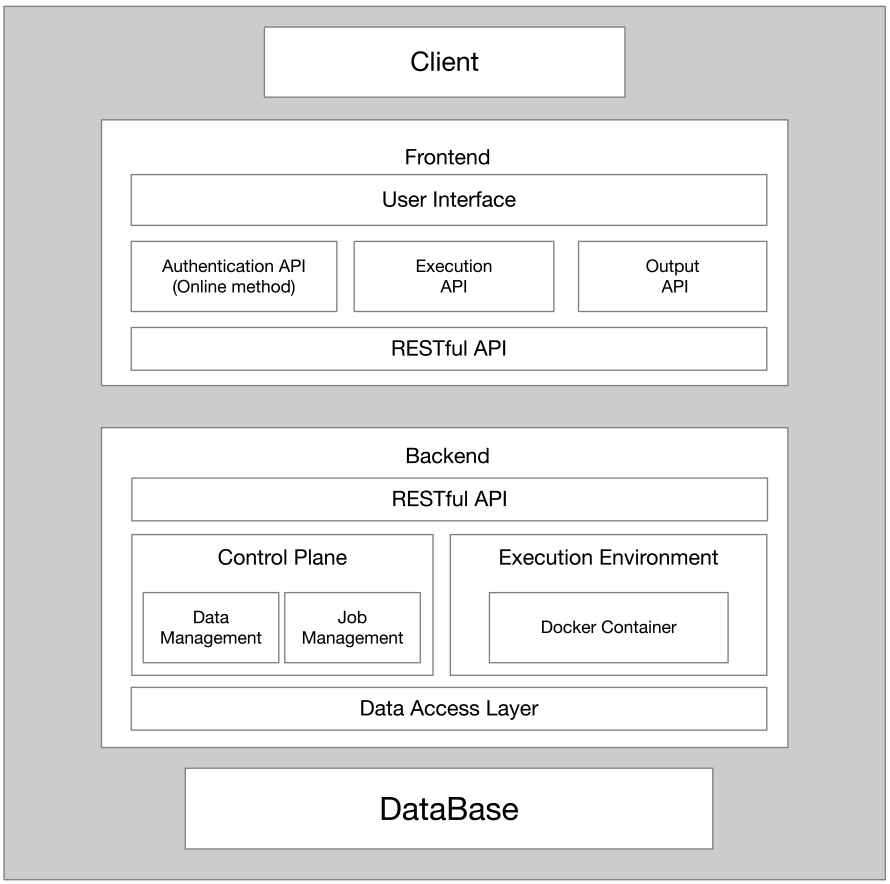
10 Scope of Work

10.1 Frontend

1. Provide UI for users to upload target exectuable, signatures files and configure parameters.

2. The parameters of pharos tools are complex and with strict limitation. A good UI may help user to set all these parameters in correct way.

3. Allow user to control and exec tools on server.



Figur 1: Module Graph

4. Allow user to observe and download any output of Pharos tools.

5. Possible techniques(popular frontend framework): Angular.js Vue.js

10.2 Backend

1. A server which could run all the Pharos tools on target exectuable.

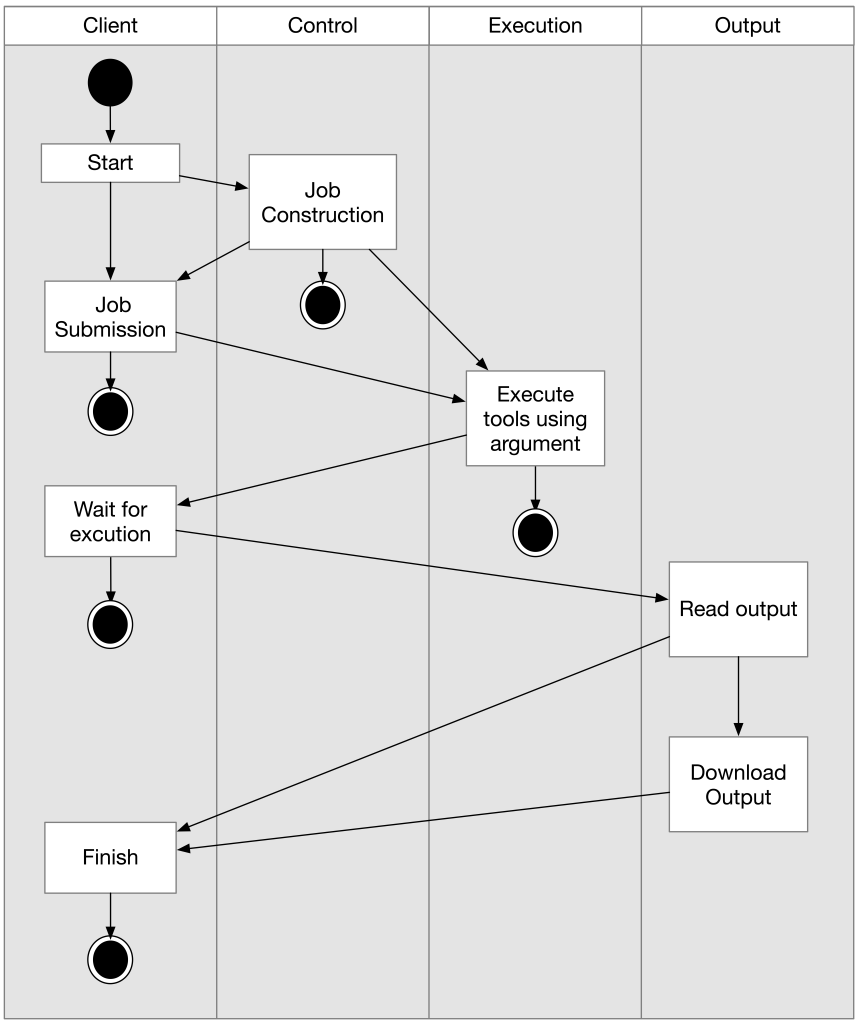
2. Save the user's target exectuable, signatures files and output of tools in a database for persistency.

3. Python-base framework(Django) would be better to support command line interface.

10.3 Database

1. The input format would be JSON and number of clients is low, a noSQL light-weighted database(MongoDB) might be better to use.

2. Providing data consistency if we have server cluster.



Figur 2: TimeSeq Graph

10.4 Dataformat

1. RESTful communication between frontend and backend to separate them for flexibility.
2. Pharos tools could output TEXT or JSON under user's demand, need to convert them to JSON before saving to database.

10.5 Dataflow

1. User-uploaded exectuable, signatures files go through backend into database.
2. User's configure parameters and backend uses Pharos tools and save output to database.
3. User look at outputs in database and download them.

11 Deliverables

The deliverables will be a prototype system to consolidate the pharos analysis tools and any/all supporting artifacts. This will include source code, test cases, design documentation, requirements documentation, and user manuals and miscellaneous documentation. If the team is successful, then there may also be an opportunity to present their work to the CERT team building the tools.

12 Schedule

Project Milestone	Deliverables	Expected Date
Sprint 0	1. Investigate the client's requirements and prioritize them 2. Draw high level architecture diagrams 3. Design the components and the architecture of the program	Feb 23rd, 2020