**Research Prototype/MVP Implementation: Quartz: Post Quantum Security of Data at Rest**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**What is the project about:**

The Quartz application scans remote targets and scan their cryptographic configurations to validate readiness of the system against post-quantum cryptographic recommendations.

**Features**

- Functional Features

* Allows the user to scan remote hosts like servers, APIs, and repositories.
* Display the host system’s post-quantum cryptographic status including the distribution of safe and unsafe cipher suites.
* Display the scan results:
  + For servers and APIs, scan results include:
    - A brief status message and a couple of data plots: cipher suite dependency plots and safe/unsafe cipher distribution
    - Detailed list of cipher suites and associated details
  + For repositories, the scan result lists the discovered components and their post-quantum safety status.
* Allow the user to add, modify, and delete algorithm information.
* Allow the user to view details of algorithms stored in the database.

- Bugs

* A bug in OpenSSL’s implementation throws an error even when a TLS connection is setup successfully.

- Test Cases:

* Invalid Inputs
  + scan type is not selected
  + scan target is not specified
  + user modifies input values mid-request
  + invalid scan target value:
    - user provides URL instead of just domain name,
    - user provides URL with path details
  + invalid scan target port or protocol value is provided
* Valid Inputs for Server Target:
  + External hosts like google.com, 142.251.214.142
  + Internal hosts like localhost
  + APIs hosted on external hosts, or virtual servers hosting services like EC2 servers (ex: ec2-12-34-567-89.us-west-2.compute.amazonaws.com)
  + Git repositories like <https://github.com/DinoTools/pysslscan>

**Design**

Functional Requirements:

The application will allow the user to:

* Scan remote targets like servers, APIs, and repositories.
* View post-quantum cryptography secure status of the system hosting the application.
  + Should also provide information on discovered ciphers and distribution of safe and unsafe components.
* View results of initiated scans against remote hosts including their post-quantum secure status and the details of cipher suites used for setting up TLS connections with the target.

System Architecture

Diagram

Description automatically generated

Components

The Quartz application has been developed using Python. The Python module can be used as a library and integrated with existing source code. It can also be called from a web server to respond to remote queries. A MySQL database server is used to store information on cipher suites and their post-quantum safety details.

There are four microservices, each designed to scan a different target type. They use different modules and open-source tools to scan the target.

* The **host scanner** module uses native SSL module to gauge the cipher suite support on the host. It performs two checks: one to determine all suites supported by the system and another to determine the cipher suites shared during connection setup. System support for cipher suites is determined using the native OpenSSL tool.
* The **server scanner** module uses the *sslscan* module to scan remote servers and return list of ciphersuites chosen by the server while setting up the connection.
* The **API scanner** uses the *testssl.sh* open-source bash script to scan remote APIs servers.
* The **repository scanner** uses a python-based open-source tool, Crypto Detector, developed by Wind River to scan remote repositories. It returns a list of components and their security status.

The application also has a dashboard that can be used to easily access the information. It has 3 components:

* Flask server: Hosts the logic behind the main application and processes user requests received from the react server.
* React server: Serves the dashboard to the user and communicates with the flask server to display information back to the user.
* MySQL server: Hosts database to store algorithm information.

APIs, Libraries and Dashboard Details

Interacting with the Library

The application can be integrated with existing code and invoked as a library.

* Client Scanner:  
  *> from pprint import pprint  
  > from microservices import client\_cipher\_suite*  
  *> cipher\_suites = client\_cipher\_suite.checkPQSafety(('google.com',443))  
  > pprint(cipher\_suites)  
  {'host': {'AES128-GCM-SHA256': 'TLSv1.2', …}, 'shared': {'AES128-GCM-SHA256': 'TLSv1.2', …}}*
* Server Scanner:  
  *> from pprint import pprint  
  > from microservices import server\_cipher\_suite  
  > cipher\_suites = server\_cipher\_suite.getServerCipherSuites('google.com')  
  > pprint(cipher\_suites)  
  {'TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA': 'TLSv12', …}*
* Repository Scanner:  
  *> from pprint import pprint  
  > from microservices import repo\_cipher\_scan  
  > components = repo\_cipher\_scan.scan\_repo(‘https://github.com/DinoTools/pysslscan’)  
  > pprint(components)  
  [[{'evidence\_type': 'generic', 'file\_path': 'https://github.com/DinoTools/pysslscan/blob/main/module/rating/builtin.py/#L19', 'matched\_text': 'cipher', 'quantum\_safe': 'No'}, …}*

Interacting with the REST API

The Flask server exposes the following endpoints:

* POST /scan: Accepts user input and scan remote targets.
  + Input format: {"scan\_type": <server|repo|api>,"target": <URL|IP>,"scan\_target\_port":"","scan\_target\_protocol":""}

Graphical user interface, text, application, email

Description automatically generated

*Fig. 1. Example of input*

* + Output format: The response from the server consists of the following details:
* graph:
  + nodes: TLS protocol versions supported by the host/client
  + edges: Cipher suites supported by each TLS protocol version and their security status
* scan\_details:
  + scan\_detail: summary of the scan
  + global quantum risk factor: Risk factor associated with the target
* scan\_results:
  + cipher suite list:
    - name
    - quantum\_safe status
    - remediation
    - risk\_factor
    - tls\_version
  + pie\_chart details: safe and unsafe colors
  + distribution of safe and unsafe components

Graphical user interface, text, application, email

Description automatically generated

* POST /listAllAlgoSpec: Returns details of all algorithms stored in the database. It takes no inputs.
  + Output format: A list of algorithm records with the following details
    - remediation
    - asymmetric encryption method
    - hash method
    - symmetric encryption method
    - name
    - pqc\_safe
    - risk\_factor

Graphical user interface, text, application

Description automatically generated

* POST /algoSpec: Inserts an algorithm record into the database if it doesn’t already exists, updates it otherwise.
  + Input format: A list of records specifying the values for each field in the algorithm table.
    - name
    - remediation
    - pqc\_safe
    - risk\_factor
    - hash method
    - asymmetric encryption method
    - symmetric encryption method

Graphical user interface, text, application, email

Description automatically generated

* + Output format: Message on status of the operation

Graphical user interface, text, application

Description automatically generated

* POST /deleteAlgoSpec: Deletes an existing algorithm record from the database.
  + Input format: A JSON object specifying the name of the algorithm to be deleted.
    - algo\_name

Graphical user interface, text, application, email

Description automatically generated

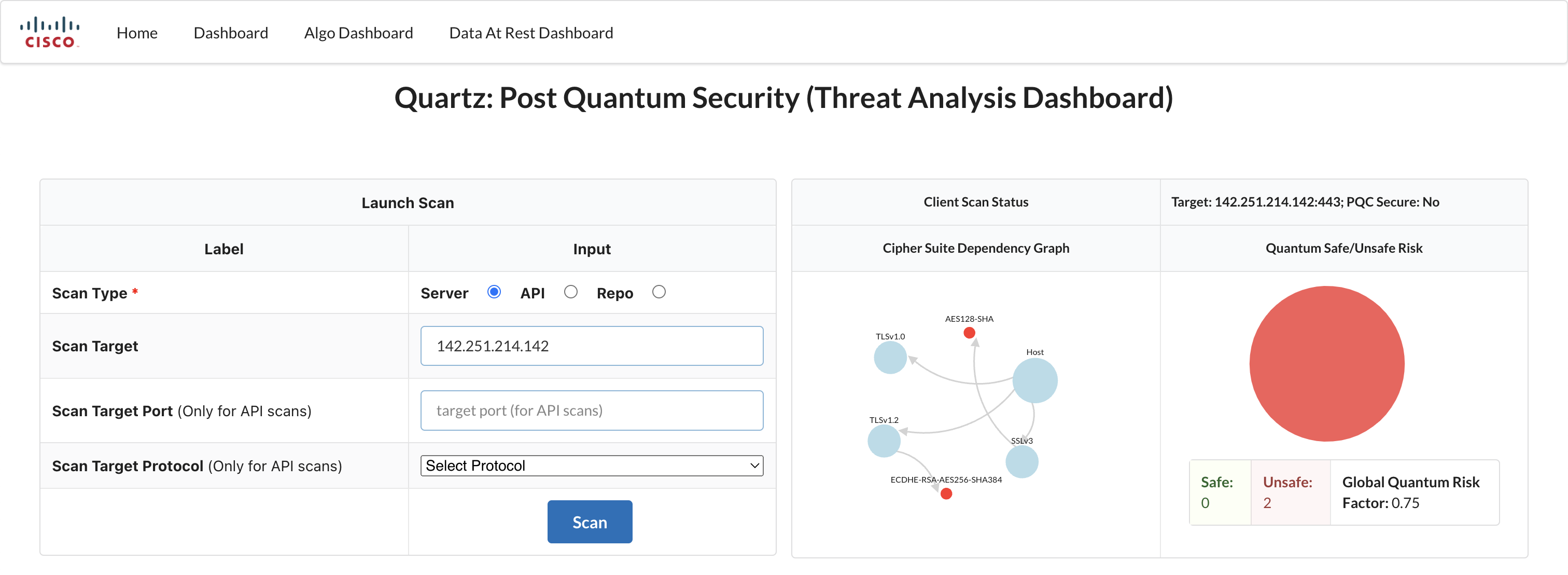
* + Output format: Message on status of the operation

Graphical user interface, text, application

Description automatically generated

Interacting with the Dashboard

Alternatively, the user can also use the front-end dashboard designed using React to access the application.



The dashboard has four input fields, of which two are mandatory. To initiate a scan, the user must select a scan type and specify a target. The port and protocol fields are required if the user wants to scan an API. Once the form is submitted, the React server requests the REST API endpoint provided by the Flask server with the user input.

The Flask server responds with details about the scan status, cipher suite dependency plot, cipher suite safe/unsafe distribution, and detailed list of cipher suites.

Graphical user interface, application

Description automatically generated

The algorithm dashboard displays an input form and a grid table displaying the existing algorithm information. The input form takes user input for an algorithm and either adds it to the database if it doesn’t exist or updates it.

Graphical user interface, table

Description automatically generated

**Algorithms**

* Host/Client Scan: The scanner performs two scans:
  + The first scan uses the native OpenSSL tool to fetch a list of all cipher suites supported by the system. Depending upon the underlying OS, the command is modified and executed using the *os* python module.
  + The second scan uses the native SSL python module to setup a TLS connection with the target and return the list of cipher suites shared by the host during the handshake process.
* Server Scan: The scanner uses the *sslscan* python module to scan the remote target. It iterates through the list of cipher suites and verifies if it is accepted by the server to setup a TLS connection.
* API Scan: The scanner executes the *testssl.sh* bash script using the target host, port, and protocol details to extract cipher suite information. A temporary file is created to store the output from the script and then deleted once the results are read.
* Repository Scan: The scanner downloads the Crypto Detector tool from its GitHub repository, if not already downloaded, and uses it to scan target repositories. The tool uses a pre-determined list of keywords to search for components and returns their list and types. The post-quantum security of each component is then checked using the evidence type against required standards.

In each scanner, the count of safe and unsafe entries is recorded and returned with the scan results. This information is used to draw pie plots depicting the distribution of each type.

Calculating the risk factor of a cipher suite

* If a cipher suite uses an unsafe method, then risk factor will be 1.
* If a cipher suite uses only safe methods, then risk factor will be equal to the maximum risk factor from among its individual components.

**How to Install**

**Cloning the repository:** Before running the dashboard, we need to fetch all the required code to our system. To clone the Git repository to the local system, run the below command.

*GitHub REPO: [Insert git clone command]*

**Launching the Dashboard:** Once the code is present on the local system, we need to deploy the application as a docker. So, we need to build the container and deploy it.

*cd quartz/quartz-dashboard*

*docker-compose build*

*docker-compose up*

**Terminating the Dashboard:** To stop the dashboard containers, type CTRL+C to stop the running instance. Then, we can stop the containers by using the below command:

*docker-compose down*

**Artefacts**

- repo: [Insert GitHub repo URL]

- design document

- demo recording