Vajra Endpoint Detection and Response tool

Backend Guide

Version 1.1

January 2024

Introduction

This documentation provides an overview of the backend architecture and implementation details for the Vajra tool used for end point detection.

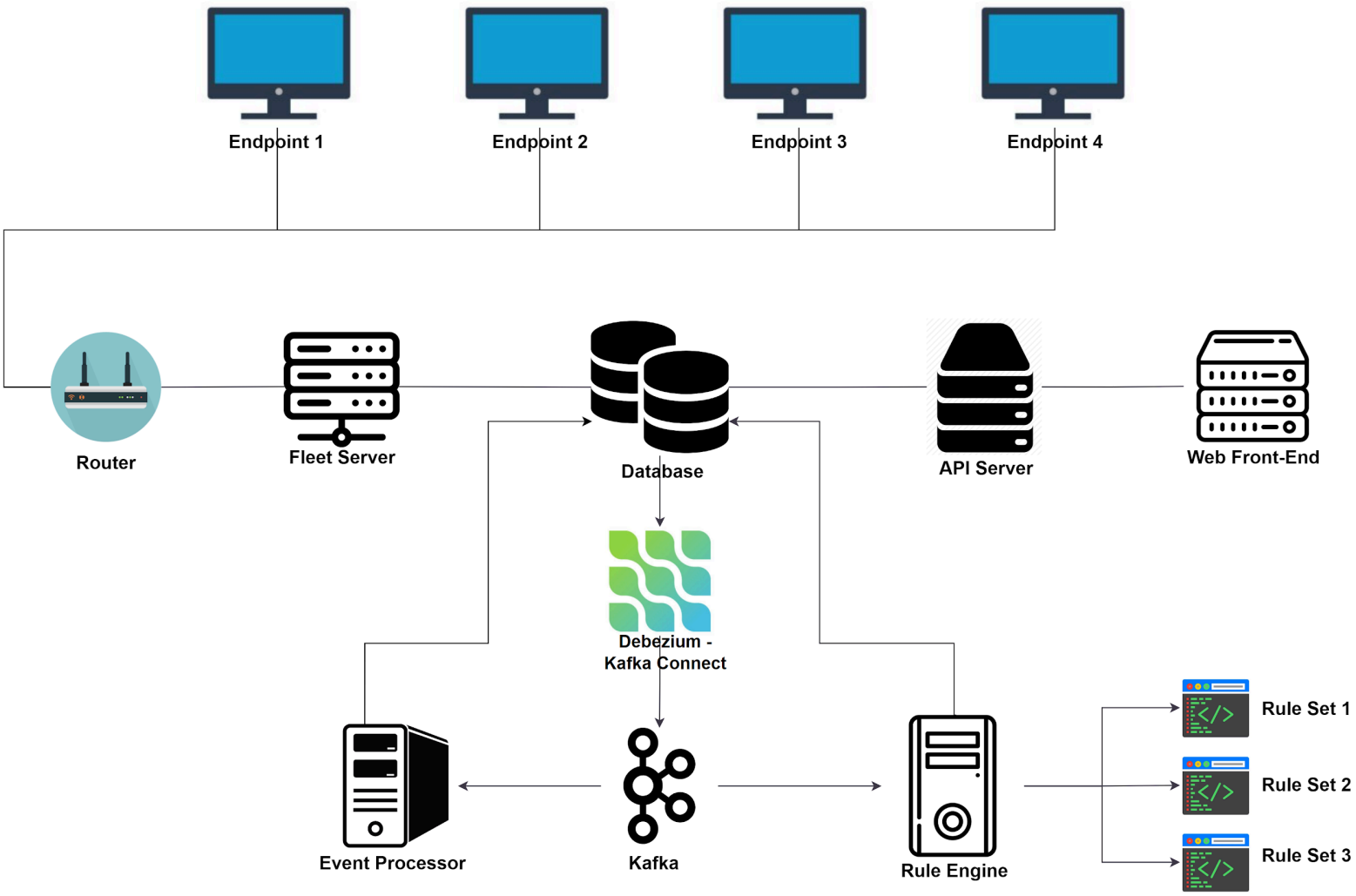
Introducing the Vajra tool, it aims to bridge this gap and assist organizations and enterprises in fortifying their security posture. Vajra offers two main functionalities:

- Identification: Vajra provides advanced capabilities to identify potential security threats within Linux and Windows-powered servers or desktops. It leverages intelligent algorithms and comprehensive scanning techniques to detect suspicious activities, unauthorized access attempts, or anomalies in system behavior.

- Investigation and Remediation: Once a threat is identified, Vajra facilitates thorough investigation and analysis. It offers a rich set of forensic tools and techniques to delve deep into the incident, enabling security teams to gather crucial evidence, determine the scope of the breach, and understand the attacker's tactics. Additionally, Vajra provides actionable remediation steps and recommendations to mitigate the impact of the attack and prevent future occurrences.

By utilizing Vajra, organizations can enhance their security posture by proactively monitoring their networks, swiftly detecting potential threats, and effectively responding to security incidents. This tool will empower organizations to stay ahead of crafty attacks executed by malicious actors, safeguard critical information, protect against financial extortion, and preserve their reputation in the face of evolving cyber threats

Architecture



The Vajra architecture consists of the following main components:

Vajra Agent

Vajra agent is powered by Osquery. Vajra agents runs on endpoints where security attacks and breaches are to be detected or blocked. All endpoints system logs to a fleet server based on predefined configuration based on a delta, i.e., only if there is

any change in collected logs, they will be forwarded to the fleet server.
Our extension of Osquery allows us to get granular information at intervals that can be customized by the user based on the requirements and resources consumed. Even if the information is gathered periodically, no event is missed as logs are 'evented'(6), i.e., every activity is captured as they happen and all the information that is captured since the last query is sent when a new query is made. We also considered the possibility that the system would restart and the PIDs would be reset, in that case also we would be able to relate the information and extract useful information.
Our system is compatible with Linux and Windows. The nicest part about our solution is that it is both agnostic to kernel versions in Linux and for windows it has full compatibility with the latest version.

Our Vajra agent can be used in stand alone mode to collect system logs for Linux and Windows systems. The Vajra agent sends logs in JSON format which can be pushed to any third party SIEM tools over HTTPS connection.

Fleet Server

We are using a custom built fleet server which can be deployed as a single binary across the environment, which helps us manage individual endpoints, database and the API server all together. This fleet server is a bare bone HTTPs server written in C++ providing maximum scalability and performance.

Database

Our application is using a relational database which provides database guarantees, enabling the data at any point to be reliable in the application. We are using postgreSQL for this purpose.

Kafka Connect - Debezium Connector

In the Scalable Log Processing architecture, the Debezium Postgres Connector is used in conjunction with Kafka Connect to enable the capture and streaming of changes from PostgreSQL databases to Kafka topics. Kafka Connect acts as the framework for managing connectors and their configurations, while the Debezium Postgres Connector is responsible for capturing and transforming the database changes.

Kafka

The architecture employs two microservices, namely the Event Processor and Rule Engine, tasked with log processing from the PostgreSQL database. Kafka bridges the gap between the database and microservices, enhancing parallelism. The system achieves increased processing speed by running multiple instances of Event Processors and Rule Engines concurrently. This parallelization is accomplished through Kafka's partitioning mechanism within the topics. Each instance reads from a distinct partition, effectively parallelizing the entire process flow. This approach optimizes log processing and bolsters the system's performance.

Read more : Scalable Log Processing Documentation

Event Processor

The event processor is the component of the system that primarily processes raw input in order to construct a process tree. The fundamental job of an event processor is to connect the dots between the processes running in the system to form a tree and to attach relevant information to the event. The process tree is concatenated to the row and isProcessed flag is set in order to indicate that the row has been processed by the event processor.

Rule Engine

Rule Engine enables EDR to analyze incoming logs which are collected from individual endpoints and send threat events which match rules to EDR as alerts. We mapped these attacks with the MITRE ATTACK matrix . These alerts can then be investigated by the analysts. They can also take appropriate action to block them in the future.

Admin Console

Admin console in UP built uses Fleet APIs and provides additional capabilities to enable analysts to easily use EDR and aid in security investigations. The UI also allows for custom Configuration, endpoint enrollment, live machine querying, and much more.

Local Machine Setup

1. **Clone the repository**

   Open a terminal and navigate to the directory where you want to store your project. Then, run the following command to clone the repository:

   git clone https://github.com/gaurishewale20/vajra_backend.git

2. **Install dependencies**

   Navigate into the project directory using the cd command:

   cd vajra_backend

   Once you're in the project directory, install the project's npm dependencies using:

   npm install

3. **PostgreSQL: Database setup**

   Install PostgreSQL: If you haven't already, install PostgreSQL on your local machine. You can download it from the official PostgreSQL website.

   CREATE DATABASE fleet;

Create a Database: Open a PostgreSQL command prompt or use a graphical tool like pgAdmin. Create a database for your project:

4. **Setting up environment variables**

We have two databases here, one for the user management and one is where the data from the fleet server will be stored. Two connections have to be established in the ORM config directory as well.

Setting up will be something as follows on the same lines:

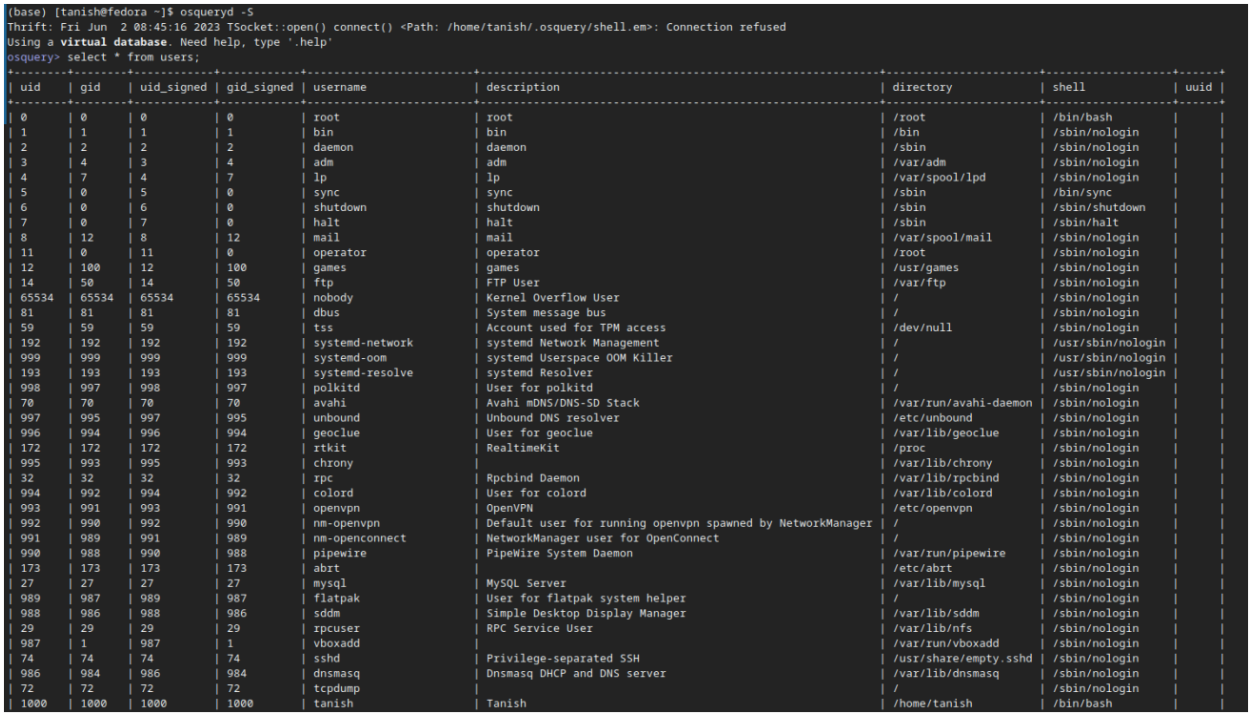
```
# Local DB 1
PG_HOST=localhost
PG_PORT=5432
POSTGRES_USER=<user-name>
POSTGRES_PASSWORD=<password>
POSTGRES_DB=<database_name>

# Fleet Server
PG_HOST1=getvajra.com
PG_PORT1=5432
POSTGRES_USER1=<user-name2>
POSTGRES_PASSWORD1=<password>
POSTGRES_DB1=fleet
```

Relevant Tools

Osquery

An operating system instrumentation, monitoring, and analytics framework called Osquery (2) gives clients' endpoints a table-like interface. It displays the operating system of the endpoint as a

high-performance relational database, enabling SQL queries to provide comprehensive, well-organized operating system data. Running processes, loaded kernel modules, active network connections, browser plugins, hardware events, file hashes, and other ideas are all represented by one of the endpoint tables. Using this information, security threats against the endpoint or endpoints can be investigated, corrected, and prevented.



```
(base) [tanish@fedora ~]$ osqueryd -S
Thrift: Fri Jun  2 08:45:16 2023 TSocket::open() connect() <Path: /home/tanish/.osquery/shell.em>: Connection refused
Using a virtual database. Need help, type '.help'
osquery> select * from users;
+-------+-------+-----------+------------+-------------------------+-----------------------------------------------------------+----------------------+-------------------+------+
| uid   | gid   | uid_signed | gid_signed | username                | description                                               | directory            | shell             | uuid |
+-------+-------+-----------+------------+-------------------------+-----------------------------------------------------------+----------------------+-------------------+------+
| 0     | 0     | 0          | 0          | root                    | root                                                      | /root                | /bin/bash         |      |
| 1     | 1     | 1          | 1          | bin                     | bin                                                       | /bin                 | /sbin/nologin     |      |
| 2     | 2     | 2          | 2          | daemon                  | daemon                                                    | /sbin                | /sbin/nologin     |      |
| 3     | 4     | 3          | 4          | adm                     | adm                                                       | /var/adm             | /sbin/nologin     |      |
| 4     | 7     | 4          | 7          | lp                      | lp                                                        | /var/spool/lpd       | /sbin/nologin     |      |
| 5     | 0     | 5          | 0          | sync                    | sync                                                      | /sbin                | /bin/sync         |      |
| 6     | 0     | 6          | 0          | shutdown                | shutdown                                                  | /sbin                | /sbin/shutdown    |      |
| 7     | 0     | 7          | 0          | halt                    | halt                                                      | /sbin                | /sbin/halt        |      |
| 8     | 12    | 8          | 12         | mail                    | mail                                                      | /var/spool/mail      | /sbin/nologin     |      |
| 11    | 0     | 11         | 0          | operator                | operator                                                  | /root                | /sbin/nologin     |      |
| 12    | 100   | 12         | 100        | games                   | games                                                     | /usr/games           | /sbin/nologin     |      |
| 14    | 50    | 14         | 50         | ftp                     | FTP User                                                  | /var/ftp             | /sbin/nologin     |      |
| 65534 | 65534 | 65534      | 65534      | nobody                  | Kernel Overflow User                                      | /                    | /sbin/nologin     |      |
| 81    | 81    | 81         | 81         | dbus                    | System message bus                                        | /                    | /sbin/nologin     |      |
| 59    | 59    | 59         | 59         | tss                     | Account used for TPM access                               | /dev/null            | /sbin/nologin     |      |
| 192   | 192   | 192        | 192        | systemd-network         | systemd Network Management                                | /                    | /usr/sbin/nologin |      |
| 999   | 999   | 999        | 999        | systemd-oom             | systemd Userspace OOM Killer                              | /                    | /usr/sbin/nologin |      |
| 193   | 193   | 193        | 193        | systemd-resolve         | systemd Resolver                                          | /                    | /usr/sbin/nologin |      |
| 998   | 997   | 998        | 997        | polkitd                 | User for polkitd                                          | /                    | /sbin/nologin     |      |
| 70    | 70    | 70         | 70         | avahi                   | Avahi mDNS/DNS-SD Stack                                   | /var/run/avahi-daemon | /sbin/nologin     |      |
| 997   | 995   | 997        | 995        | unbound                 | Unbound DNS resolver                                      | /etc/unbound         | /sbin/nologin     |      |
| 996   | 994   | 996        | 994        | geoclue                 | User for geoclue                                          | /var/lib/geoclue     | /sbin/nologin     |      |
| 172   | 172   | 172        | 172        | rtkit                   | RealtimeKit                                               | /proc                | /sbin/nologin     |      |
| 995   | 993   | 995        | 993        | chrony                  |                                                           | /var/lib/chrony      | /sbin/nologin     |      |
| 32    | 32    | 32         | 32         | rpc                     | Rpcbind Daemon                                            | /var/lib/rpcbind     | /sbin/nologin     |      |
| 994   | 992   | 994        | 992        | colord                  | User for colord                                           | /var/lib/colord      | /sbin/nologin     |      |
| 993   | 991   | 993        | 991        | openvpn                 | OpenVPN                                                   | /etc/openvpn         | /sbin/nologin     |      |
| 992   | 990   | 992        | 990        | nm-openvpn              | Default user for running openvpn spawned by NetworkManager | /                    | /sbin/nologin     |      |
| 991   | 989   | 991        | 989        | nm-openconnect          | NetworkManager user for OpenConnect                       | /                    | /sbin/nologin     |      |
| 990   | 988   | 990        | 988        | pipewire                | PipeWire System Daemon                                    | /var/run/pipewire    | /sbin/nologin     |      |
| 173   | 173   | 173        | 173        | abrt                    |                                                           | /etc/abrt            | /sbin/nologin     |      |
| 27    | 27    | 27         | 27         | mysql                   | MySQL Server                                              | /var/lib/mysql       | /sbin/nologin     |      |
| 989   | 987   | 989        | 987        | flatpak                 | User for flatpak system helper                            | /                    | /sbin/nologin     |      |
| 988   | 986   | 988        | 986        | sddm                    | Simple Desktop Display Manager                            | /var/lib/sddm        | /sbin/nologin     |      |
| 29    | 29    | 29         | 29         | rpcuser                 | RPC Service User                                          | /var/lib/nfs         | /sbin/nologin     |      |
| 987   | 1     | 987        | 1          | vboxadd                 |                                                           | /var/run/vboxadd     | /sbin/nologin     |      |
| 74    | 74    | 74         | 74         | sshd                    | Privilege-separated SSH                                   | /usr/share/empty.sshd | /sbin/nologin     |      |
| 986   | 984   | 986        | 984        | dnsmasq                 | Dnsmasq DHCP and DNS server                               | /var/lib/dnsmasq     | /sbin/nologin     |      |
| 72    | 72    | 72         | 72         | tcpdump                 |                                                           | /                    | /sbin/nologin     |      |
| 1000  | 1000  | 1000       | 1000       | tanish                  | Tanish                                                    | /home/tanish         | /bin/bash         |      |
```

Technology Stack

This section details the essential components of the technology stack needed to develop the API server responsible for retrieving data from the database and facilitating the display of frontend features. The selection of these technologies plays a crucial role in ensuring efficient communication between the backend and frontend of the application.

Node.js

Node.js is a powerful and efficient server-side JavaScript runtime that allows you to build scalable and high-performance applications. It uses an event-driven, non-blocking I/O model, making it well-suited for handling asynchronous operations and real-time applications. In our project, Node.js will serve as the foundation of our API server, responsible for retrieving data from the database and facilitating communication with the frontend.

PostgreSQL

PostgreSQL is a powerful open-source relational database management system (RDBMS) that offers robust data storage, querying capabilities, and support for complex transactions. It will play a critical role in our application's backend by storing and managing the data that our API server interacts with.

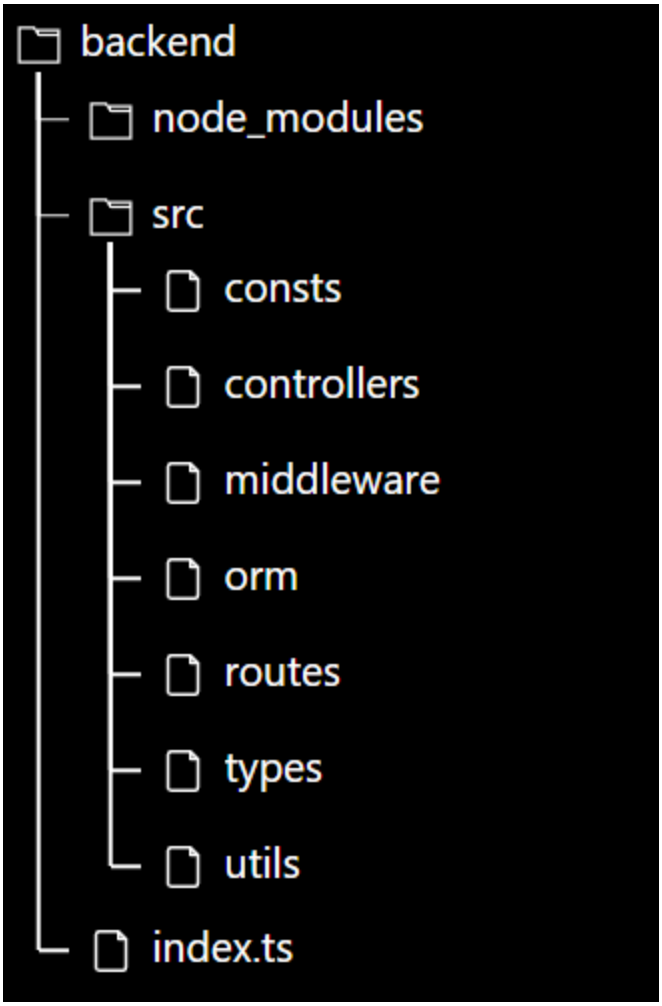
Typescript

TypeScript is a superset of JavaScript that adds static typing and other advanced features to the language. It enhances the development process by providing compile-time type checking, improved code organization, and better tooling support. TypeScript will be a key component in our project's frontend and backend codebase.

Project Structure

Directory Structure

Below is the high-level directory structure of the Website API backend project.



Directory Descriptions

/src

The /src directory is the heart of our backend application, housing the source code responsible for handling various aspects of the application's functionality. This directory is organized into subdirectories that represent different layers and components of the backend architecture.

/consts

The /consts directory contains modules that define constants, configuration settings, and other global values that are used throughout our backend application. By centralizing these values in one location, we promote consistency, maintainability, and easier updates across the codebase.

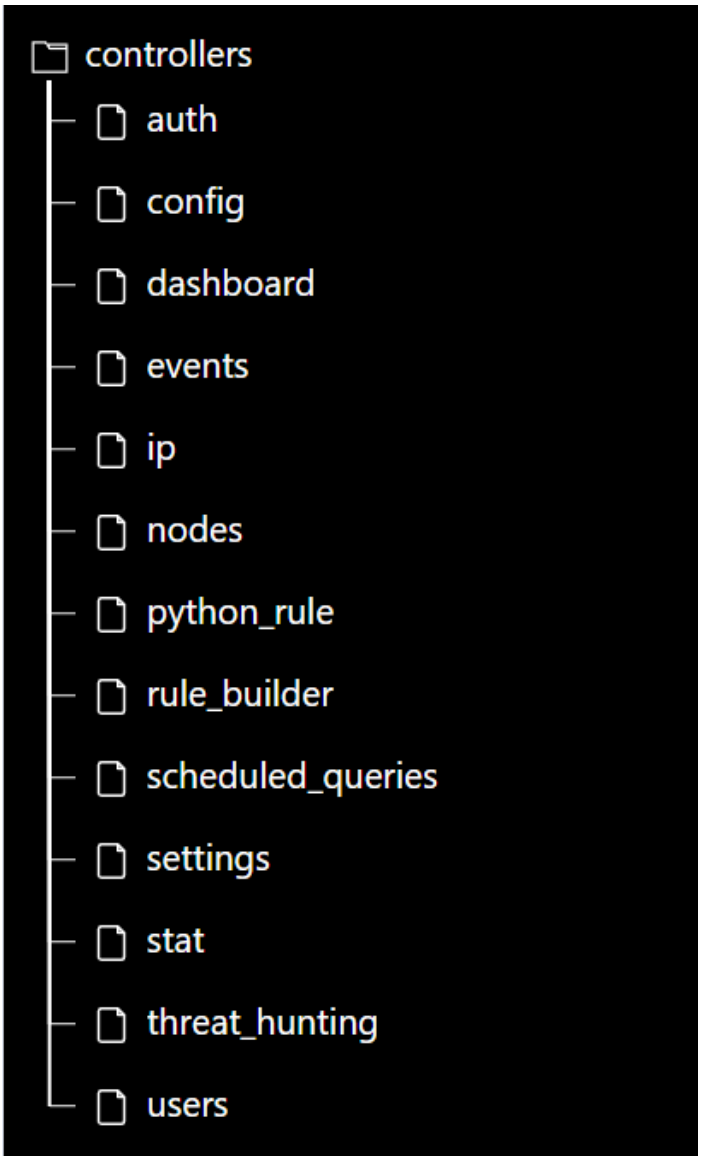
Example:

```
export enum ConstsUser {
  PASSWORD_MIN_CHAR = 4,
}
```

/controllers

The /controllers directory contains modules responsible for handling incoming HTTP requests, processing data, and orchestrating the business logic of the application. Each controller typically corresponds to a specific set of related endpoints and interacts with the appropriate models to retrieve or manipulate data.

Further bifurcation of the controllers module involves the following modules:



- **/auth:**

  This module contains functions related to authentication like login, register, renew_access_token, etc.

- **/config:**

  This module contains functions related to configuration management such as addition, deletion, updation, and listing of configuration queries.

- **/dashboard:**

  This module returns data for populating different graphs on the dashboard. For example, alerts count, events count, weekly alert updates, etc.

- **/events:**

  This module contains functions for listing, updating, querying events and creating process trees.

- **/ip:**

  //TODO: completion left

- **/nodes:**

  This module contains functions for listing nodes, configuration of nodes, querying and memory_health of nodes, etc.

- **/python_rule:**

  This module contains functions related to python rules management such as addition, deletion, updation, and listing of python rules.

- **/rule_builder:**

  This module contains functions related to rules management such as addition, deletion, updation, and listing of rules.

- **/scheduled_queries:**

  This module contains functions for scheduling live queries on nodes, sending their status, and response of particular scheduled queries.

- **/settings:**

  This module contains functions related to manual data purging and data retention.

- **/threat_hunting:**

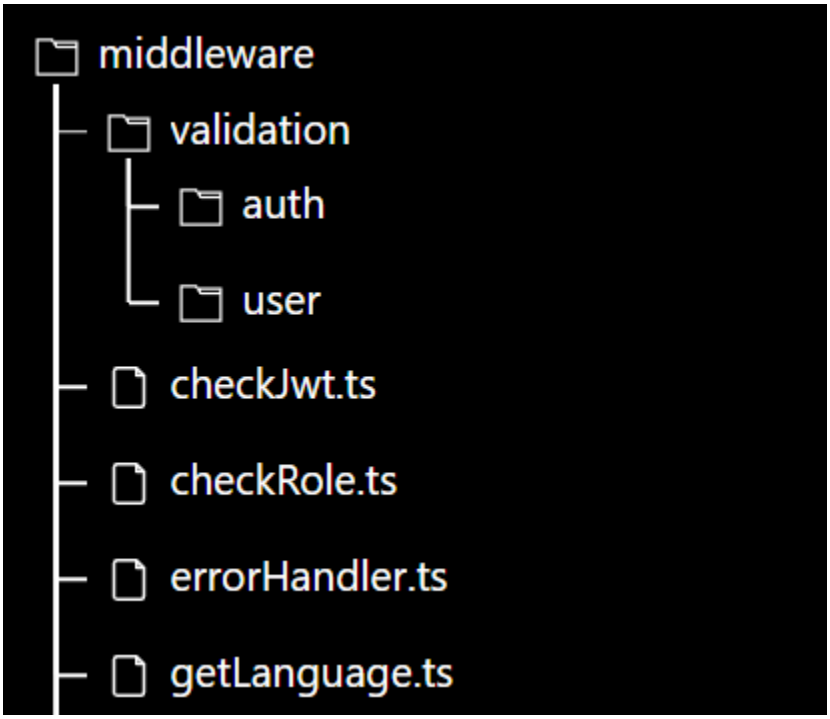
  This module contains functions for handling techniques and sub techniques.

- **/users:**

  This module contains functions related to user management like creating teams, editing permissions, assigning permissions, listing teams and admins, etc.

/middleware

The /middlewares directory contains modules that implement middleware functions. Middleware functions sit between the client's request and the final endpoint handler, performing tasks like authentication, input validation, and logging.



/orm

In the /orm directory, you'll find modules that define the ORM entities, relationships, and database mappings used in our application. ORM entities encapsulate the logic required to interact with the database, ensuring data integrity and streamlining database operations.

Example:
Configuration connecting to the Postgresql database:

```
const config1: DataSourceOptions = {
  type: 'postgres',
  name: 'default',
  host: process.env.PG_HOST1,
  port: Number(process.env.PG_PORT1),
  username: process.env.POSTGRES_USER1,
  password: process.env.POSTGRES_PASSWORD1,
  database: process.env.POSTGRES_DB1,
  synchronize: false,
  logging: false,
```

```
  entities: ['src/orm/entities/DB2/**/*.ts'],
  migrations: ['src/orm/migrations/**/*.ts'],
  subscribers: ['src/orm/subscriber/**/*.ts'],
  namingStrategy: new SnakeNamingStrategy(),
};
```

Entities containing data models:

```
@Entity('users')
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({
    unique: true,
  })
  email: string;

  @Column()
  password: string;

  @Column({
    nullable: true,
    unique: true,
  })
  username: string;

  @Column({
    nullable: true,
  })
  name: string;

  @Column({
    default: 'STANDARD' as Role,
    length: 30,
  })
  role: string;

  @Column({
    default: 'en-US' as Language,
    length: 15,
```

```
  })
  language: string;

  @Column()
  @CreateDateColumn()
  created_at: Date;

  @Column()
  @UpdateDateColumn()
  updated_at: Date;

  setLanguage(language: Language) {
    this.language = language;
  }

  hashPassword() {
    this.password = bcrypt.hashSync(this.password, 8);
  }

  checkIfPasswordMatch(unencryptedPassword: string) {
    return bcrypt.compareSync(unencryptedPassword, this.password);
  }
}
```

/routes

The /routes directory contains modules that define the API routes and map them to the appropriate controller methods. Each route module aggregates related endpoints, making it easier to manage and maintain our API.

Example:

```
const router = Router();

router.get("/table_name", [checkJwt, checkRole(['ADMINISTRATOR', 'STANDARD'],
true)], table_list);
router.get("/list", [checkJwt, checkRole(['ADMINISTRATOR', 'STANDARD'], true)],
config_list);
router.post("/add", [checkJwt, checkRole(['ADMINISTRATOR', 'STANDARD'], true)],
config_add);
```

```
router.post("/update", [checkJwt, checkRole(['ADMINISTRATOR', 'STANDARD'], true)],
config_update);
router.delete("/delete", [checkJwt, checkRole(['ADMINISTRATOR', 'STANDARD'], true)],
config_delete);

export default router;
```

/types

This folder stores the format of certain objects.

Example:

```
import { Role } from '../middleware/orm/entities/DB1/users/types';

export type JwtPayload = {
  id: number;
  name: string;
  email: string;
  role: Role;
  created_at: Date;
};
```

/utils

In the /utils directory, you'll find utility modules that contain reusable functions used across different parts of the backend application. These utilities can help streamline code and avoid duplication.

Example:

```
import jwt from 'jsonwebtoken';

import { JwtPayload } from '../types/JwtPayload';
import { RenewAccessJwtPayload } from 'types/RenewAccessJwtPayload';

const jwtTokenGenerator = (payload : any): string =>{
  return jwt.sign(payload, process.env.JWT_SECRET!, {
    expiresIn: process.env.JWT_EXPIRATION,
  });
}
export const createJwtToken = (payload: JwtPayload): string => {
  return jwtTokenGenerator(payload);
```

```
};

export const createRenewAcessJwtToken = (payload: RenewAccessJwtPayload): string =>{
  return jwtTokenGenerator(payload);
}
```

index.ts

Initializes the server, sets up middleware, and listens on a specified port for incoming requests.

Example:

```
export const app = express();
app.use(cors());
app.use(helmet());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(getLanguage);

try {
  const accessLogStream = fs.createWriteStream(path.join(__dirname,
'../log/access.log'), {
    flags: 'a',
  });
  app.use(morgan('combined', { stream: accessLogStream }));
} catch (err) {
  console.log(err);
}
app.use(morgan('combined'));

app.use('/', routes);

app.use(errorHandler);

const port = process.env.PORT || 4000;
app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});

(async () => {
```

```
  dataSource = await dbCreateConnection(config1, config2);
})();

export const dbSourceConnection=()=>{
  return dataSource;
}
```