VULCAN

Rule-Based Vulnerability Detection

A problem we often encounter in the real world is trying to understand which of our servers are vulnerable according to a reference vulnerabilities registry and then generating alerts by matching specified rules - this is very similar to what the standard antivirus software applications do on your own devices.

Let's try to implement such a mechanism and build a detection pipeline that can be used periodically and help us keep our servers safe.

Important:

- This task refers to the directory you write your solution in as: `<SOLUTION_DIR>`
- Read the task and make sure you understand it completely - the data in play, the desired flow, the expected result, etc. If you have any questions, feel free to discuss them with your interviewer before you begin.
- We are expecting your solution to be production ready - make sure you are confident in your delivery (clarity and readability, time and space efficiency, error handling, comments, etc.).
- Your solution should be robust enough to handle real-life situations, as well as be flexible enough to allow for future changes in requirements - features, scope, scale, etc. Stay within the scope of the time you are allotted, but be ready to discuss potential changes and enhancements during the review.
- In particular - assume all relevant data can fit in memory (i.e. no need to use databases, distributed processing, etc.).
- If you need to make assumptions or compromises, make such decisions clearly and be ready to explain them.
- Inspect the data you are working with - it can be varied, incomplete or duplicate; make sure you are handling the edge cases correctly.
- Feel free to use whatever objects and libraries you need - as long as they do not trivialize significant portions of the task. **Note:** there is no need for powerful frameworks (e.g. Django or Spring), but you can use whatever tools you are comfortable with.
- Your interviewer will check upon you from time to time to see if all proceeds well, but if at any point you feel stuck or something is not clear, do not hesitate to contact the interviewer - we want you to do your best!

Good luck!

Attention: there is a detailed example of the full process at the end - do not skip it!

---

1. Servers

First, we get a list of the relevant servers we have from:
`http://interview.vulcancyber.com:3000/servers`

Making a GET request there with HTTP header of "`Authorization: Aa123456!`" will return a JSON of the following structure:

```
[
  {
    "id": integer,
    "hostname": string,
    "ip": string,
    "os": string,
    "osVersion": string
  },
  ...
]
```

2. Vulnerabilities

Next, we need to get the data from a reference vulnerabilities registry at:
`http://interview.vulcancyber.com:3000/vulns`

Making a POST request there with a body in the format:
`{"startId": integer starting at 1, "amount": integer}`
will return a JSON of the following structure with the requested `amount` of vulnerabilities starting at `startId`:

```
[
  {
    "id": integer,
    "name": string,
    "risk": float,
    "affects": string of the format "os_osVersion"
  },
  ...
]
```

We consider a vulnerability to affect a server when both the OS and the version of the server match the 'affects' field of the vulnerability.

3. <u>Detection rules</u>
Finally, let's get the detection rules we will use to trigger alerts on a successful match. The CSV file can be downloaded from:
`http://interview.vulcancyber.com:3000/rules.csv`
Download it to a directory of your choice (preferably - `<SOLUTION_DIR>`).

Each row in the file ***after the title row*** follows the following format:
`type,parameter,operator,value`

where:
- `type` can be one of: `server, vulnerability`
- `parameter` can be any field in the server and vulnerability JSON objects
- `operator` can be one of: `eq, gt, lt` - for equals, greater than and lesser than, accordingly
- `value` can be either a number or a string, according to the parameter targeted

When ***ALL*** the rules are triggered at once we count that as a detection that should generate an alert.

Note: if there are more than one rule, matching only one of the rules is NOT counted as a detection - it should be a logical AND between ALL the rules in the file

---

4. <u>Alerts:</u>
Our ultimate goal is to output the detections from the previous step for future processing by other systems (or human eyes).

For each detection append a line of the following format to a log file at
`<SOLUTION_DIR>/detections.log`:

```
vulnerability <VULNERABILITY_NAME> with risk <RISK> discovered on
<HOSTNAME> <IP>
```

where `VULNERABILITY_NAME` corresponds to the vulnerability `"name"` field, `HOSTNAME` to server `"hostname"` field, `IP` to server `"ip"` field and `RISK` to vulnerability `"risk"` field.

Example:

Assuming we get the following servers for **step 1**:

```
[
  {
    "id": 1253612,
    "hostname": "primary_dns_server",
    "ip": "23.2.1.123",
    "os": "CentOS",
    "osVersion": "1.12_33"
  },
  {
    "id": 2373487,
    "hostname": "primary_file_storage",
    "ip": "3.12.230.5",
    "os": "WindowsServer",
    "osVersion": "2012R2"
  }
]
```

And the following vulnerabilities for **step 2** when we send with the body as `{"startId": 1, "amount": 10}`:

```
[
  {
    "id": 1,
    "name": "Misconfigured user permissions mechanism",
    "risk": 8.0,
    "affects": "CentOS_1.12_33"
  },
  {
    "id": 2,
    "name": "Privileged decompression bombing",
    "risk": 5.35,
    "affects": "WindowsServer_2012R2"
  }
]
```

Observe: getting only 2 results when asking for 10 means we're at an end.

With the following rules in **step 3**:

```
type,parameter,operator,value
server,os,eq,CentOS
vulnerability,risk,gt,7
```

We would expect the following output in the log file in **step 4**:

```
vulnerability Misconfigured user permissions mechanism with risk 8.0 discovered
on primary_dns_server 23.2.1.123
```