RSA

GUC
German University in Cairo

Systems and Network Security

Comprehensive Report

Milestone 4

Group Assignments: Group 4: Simulate 10 requests per second for 2 minutes (as per last update)

OWN YOUR IDENTITY.

Task

In this milestone, you will perform stress testing using Apache JMeter to evaluate the performance of the Cloud Authentication Service (CAS) in handling REST API requests and the Identity Router (IDR) in processing RADIUS authentications. The goal is to analyze how these systems perform under different loads and document findings in a structured report.

Task Overview:

Each team will simulate different load scenarios using Apache JMeter and observe system behavior under different request rates. The requests will be directed separately to CAS for REST API testing and IDR for RADIUS authentication testing, based on the assigned group tasks outlined below. The students will then analyze the performance metrics and provide a detailed report on their methodology, observations, and conclusions.

Testing Requirements:

- REST API Performance Test: Simulate API requests sent to CAS URL (HTTPS requests) and measure response times, error rates, and system behavior underload.
- RADIUS Authentication Test: Simulate authentication requests to IDR using the RADIUS module in JMeter and analyze performance metrics such as authentication time and success rate.

Submission Requirements:

Each team must submit two documents:

1. **Comprehensive Report:**
    - Test Plan & Configuration: Steps taken to configure JMeter for REST API and RADIUS tests including JMeter configuration and any commands used.
    - Conclusion & Recommendations: Insights gained from the tests and potential areas for optimization.
2. **Summary report and dashboard:**
    - A dashboard and a report generated by JMeter after running the tests containing numerical data, visualizations, and charts illustrating key performance indicators such as response time, transactions per second (TPS), error rate, and system stability trends.

This is the drive link for all the required submissions

- Tests (.jmx)
- Html Dashboards (.html)
- Summary report (.jtl)
- This **Comprehensive Report.**

https://drive.google.com/drive/folders/1z0qzD7wsnz960aiscMNFU_Ljdo8fFeuC?usp=drive_link

DISCLAIMER: The requirements for our group was changed from 40 req/sec to 10 req/sec

Test Plan & Configuration:

Requirements

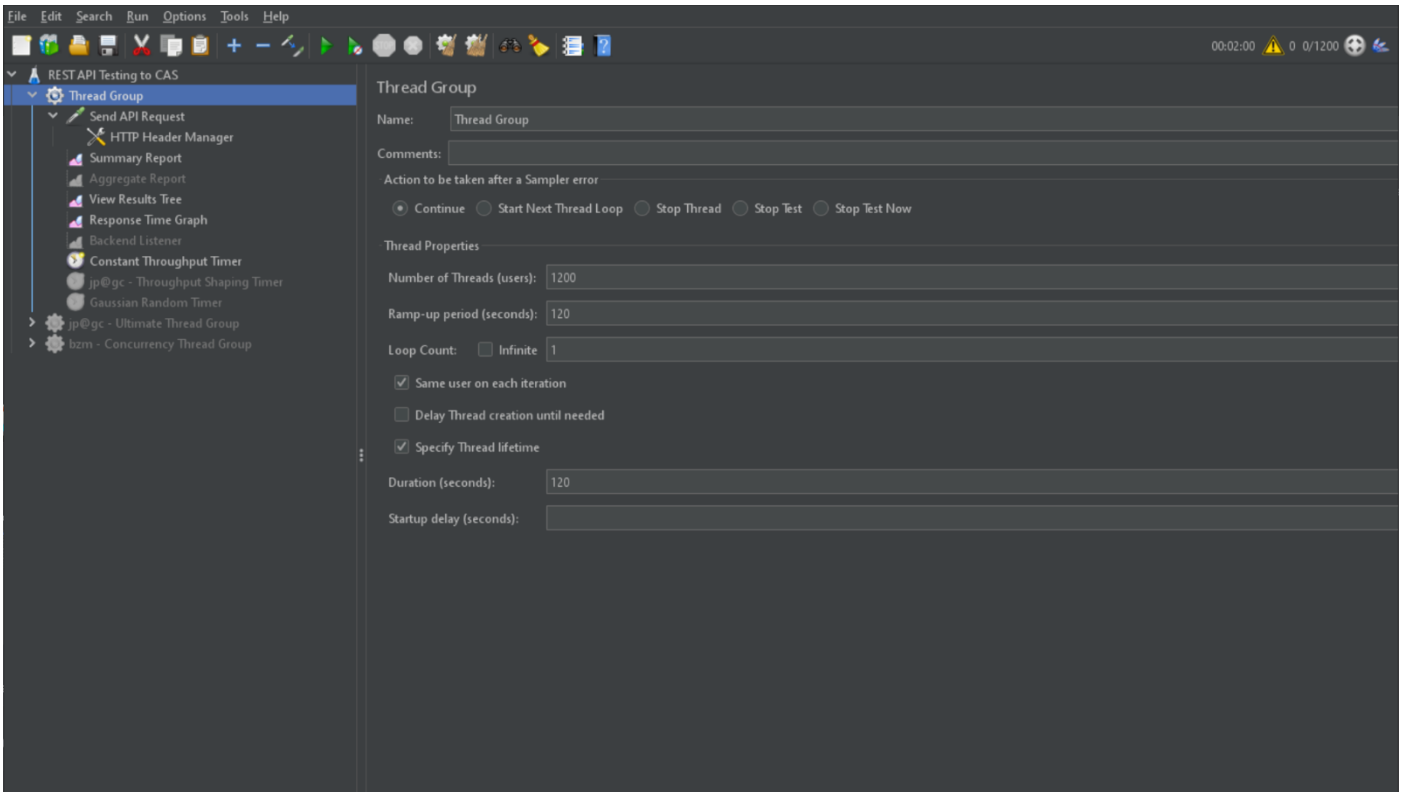
- Download JMeter from here Apache JMeter - Download Apache JMeter
- Download Java from here Download Java

REST API Performance Test

Step 1: Open JMeter and Create a Test Plan

1. Open **Apache JMeter**.
2. Right-click on **Test Plan** → Add → **Threads (Users)** → **Thread Group**.

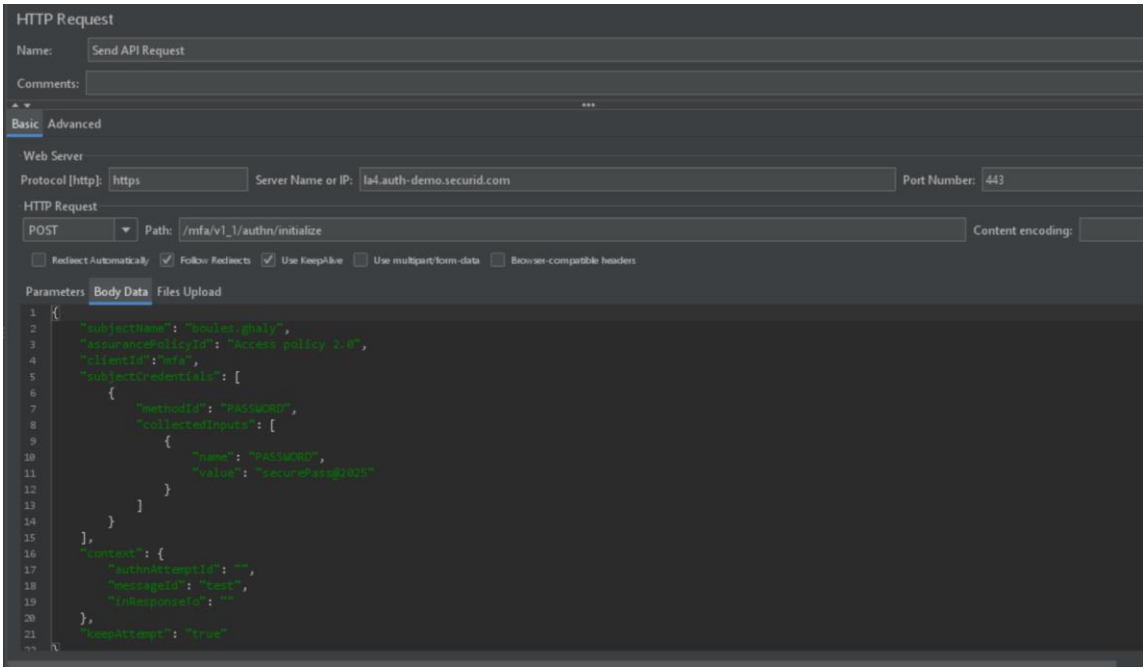
Step 2: Configure the Thread Group



- Number of Threads (users): `1200`
- Ramp-Up Period: `120` (means 10 threads/sec [1200/120 = 10])
- Loop Count: `1`
- Duration: `120`

Step 3: Add an HTTP Request Sampler

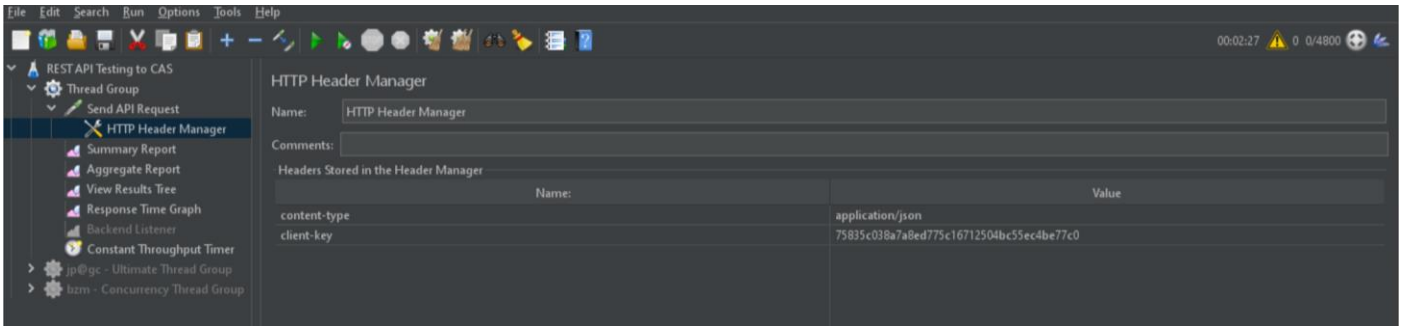
Right-click on the **Thread Group** → Add → **Sampler** → **HTTP Request**



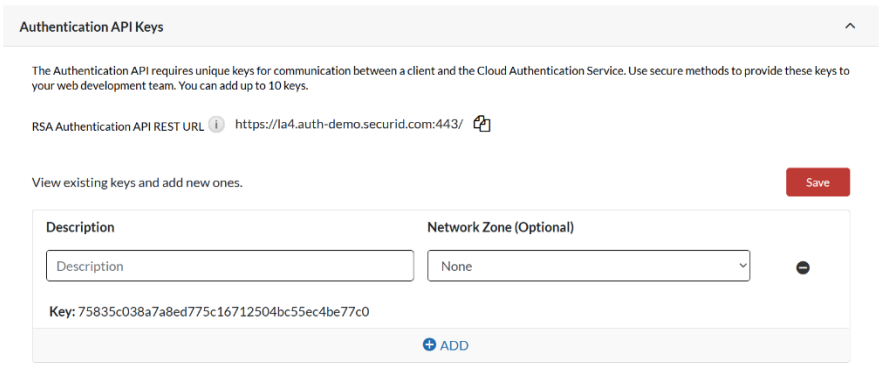
- Name: `Send API Request`
- Server Name or IP: (`la4.auth-demo.secureid.com`)
- Protocol: `https`
- Path: `/mfa/v1_1/authn/initialize`
- Method: `POST`
- Body Data

Step 4: Add HTTP Header Manager

Right-click on the **HTTP Request** → Add → **Config Element** → **HTTP Header Manager**

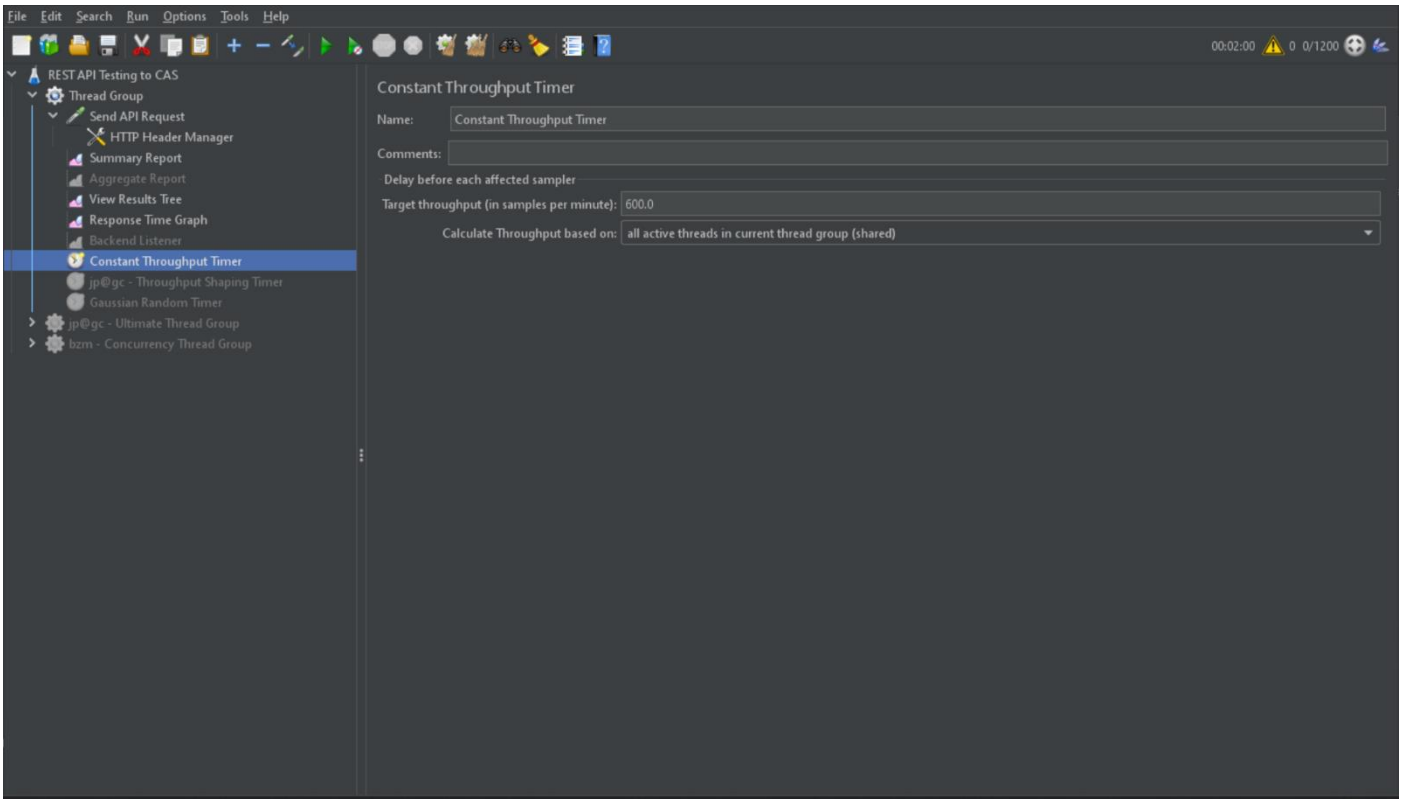


- Add headers like:
  - `Content-Type: application/json`
  - `Client-key:`



Step 5: Add Constant Throughput Timer

- Right-click on the **Thread Group** → Add → **Timer** → **Constant Throughput Timer**
- **This is to maintain a constant throughput for the required requests/sec**



Step 6: Add Listeners (for Results)

Add the following:

- **Summary Report**
- **View Results Tree**
- **Response Time Graph**
- **Backend Listener** (for dashboard) only active when run from CMD

Step 7: Save and Run the Test

Step 8: Generate the HTML Dashboard (After the test)

1. Open the terminal from `jmeter.bat` directory (bin)
2. Use this command in terminal:

```
jmeter.bat -n -t ..\..\CAS_TEST.jmx -l ..\..\results.jtl -e -o ..\..\dashboard
```

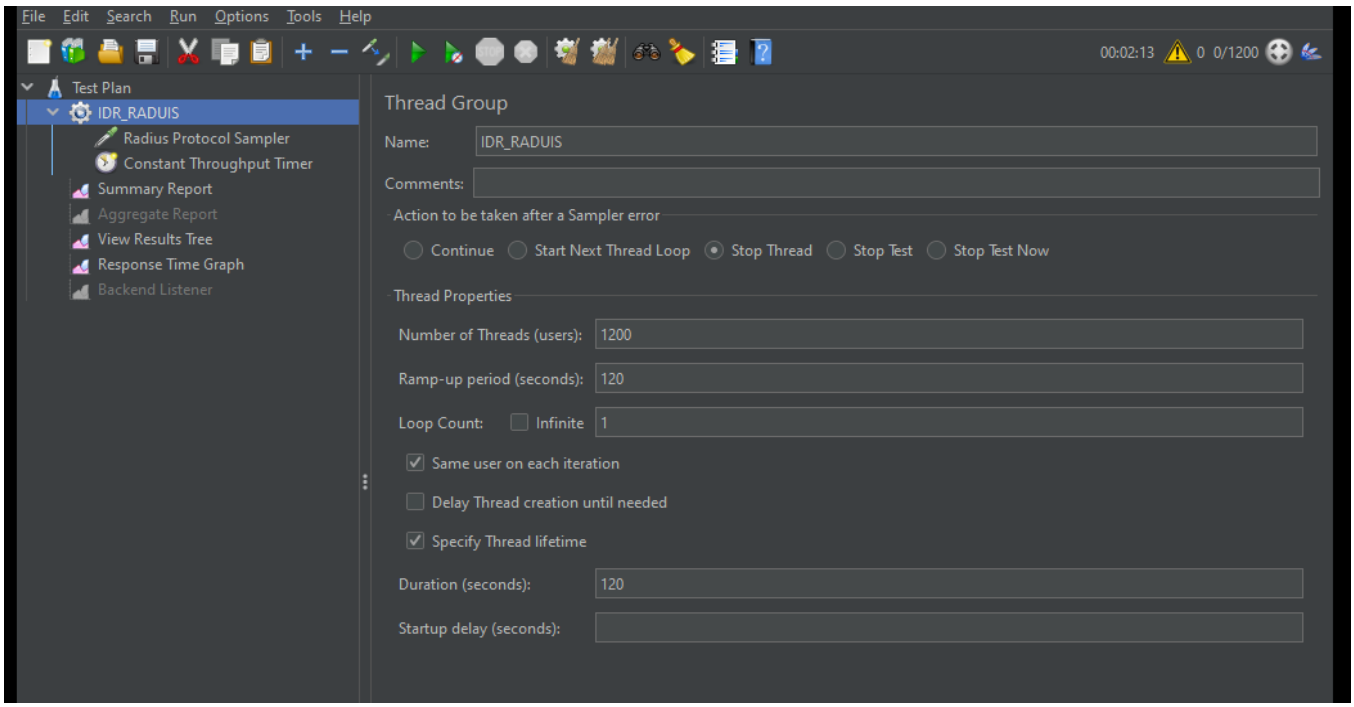
3. Open the generated dashboard `dashboard/index.html` to view charts and metrics.

Radius Authentication Test

Step 1: Install RADIUS Plugin

1. Download `JMeterRadiusSampler.jar` plugin and place it in the `lib/ext` directory of JMeter.

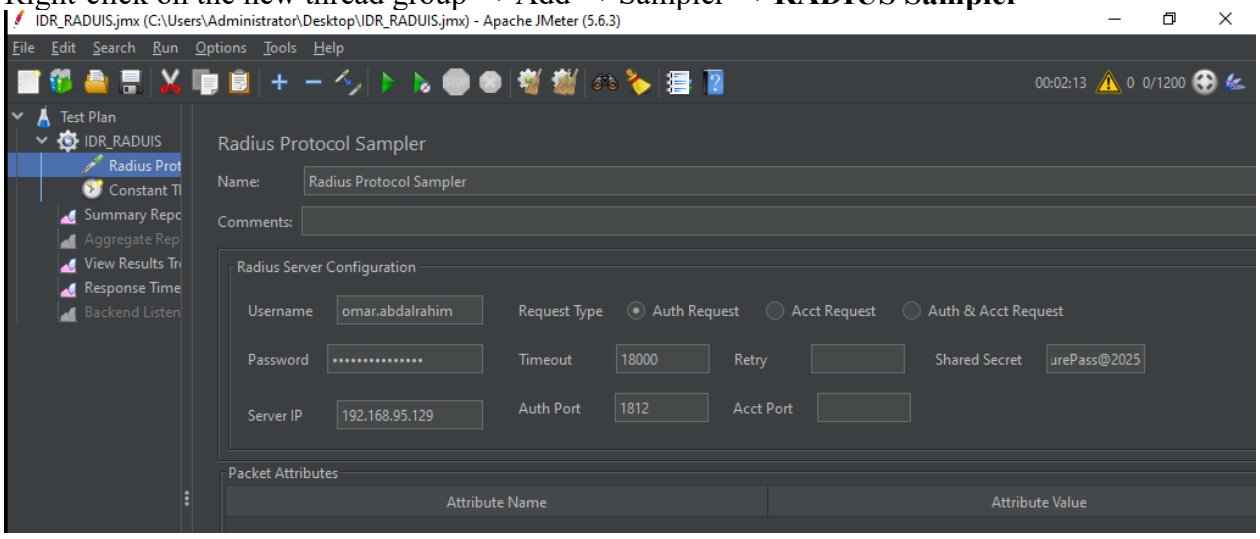
Step 2: Create a New Thread Group



- Number of Threads: `1200`
- Ramp-Up: `120`
- Loop Count: `1`
- Duration: `120`

Step 3: Add RADIUS Sampler

Right-click on the new thread group → Add → Sampler → **RADIUS Sampler**



- Server: IP address of the IDR
- Port: `1812` (default RADIUS port)
- Shared Secret: securePass@2025
- User Name: omar.abdalrahim (any user from the AD)
- Password: securePass@2025
- Timeout: 18000 (ms)

Step 4: Add Constant Throughput Timer

- Right-click on the **Thread Group** → Add → **Timer** → **Constant Throughput Timer**
- **This is to maintain a constant throughput for the required requests/sec**

Step 5: Add Listeners (Same as before)

- Summary Report
- View Results Tree
- Response Time Graph
- Backend Listener (for dashboard) only active when ran from CMD

Step 6: Save and Run

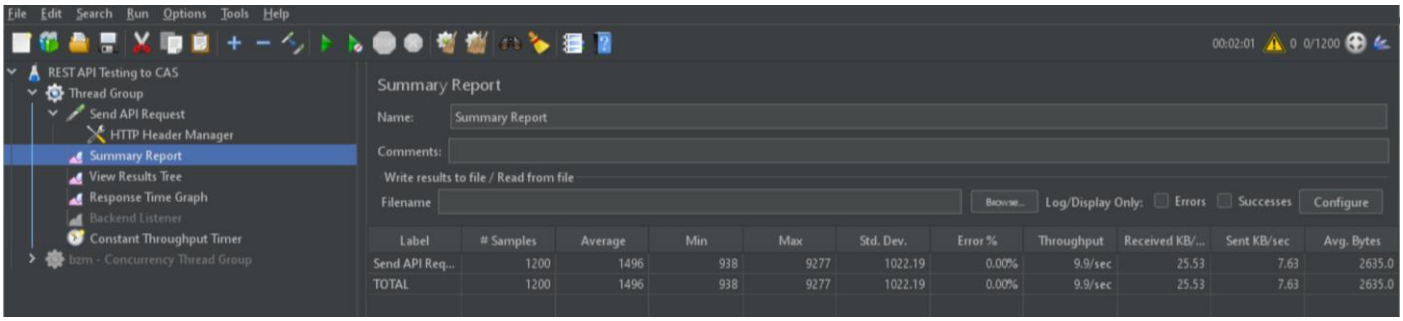
Step 7: Generate HTML Dashboard

1. Open the terminal from `jmeter.bat` directory (bin) and run the following command

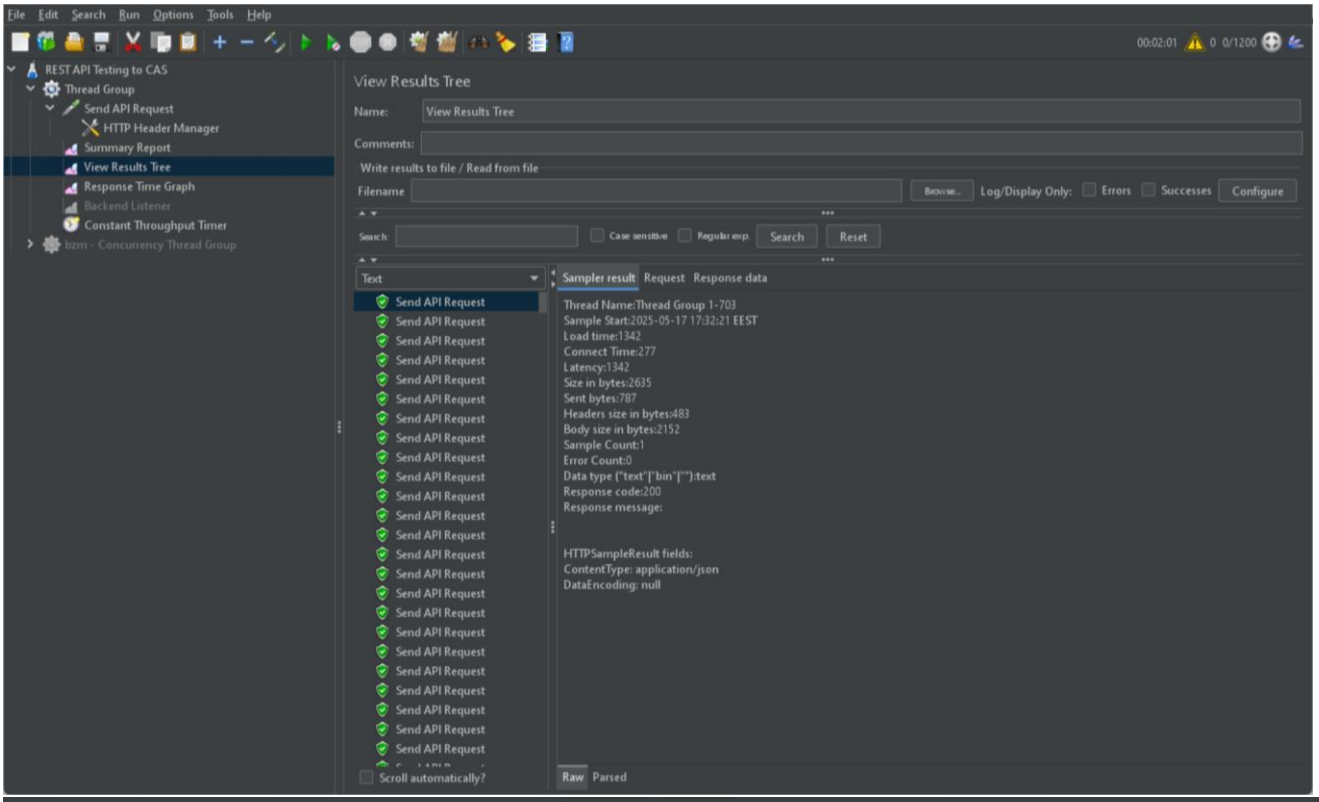
```
jmeter.bat -n -t ..\..\IDR_Test.jmx -l ..\..\results.jtl -e -o ..\..\dashboard
```

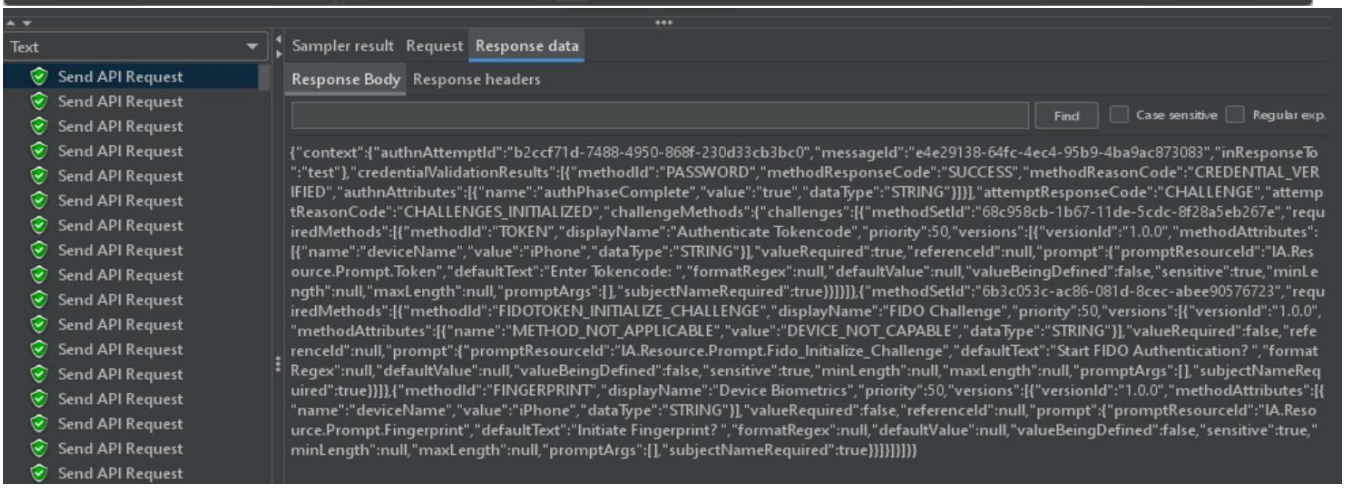
Conclusion:

CAS_TEST

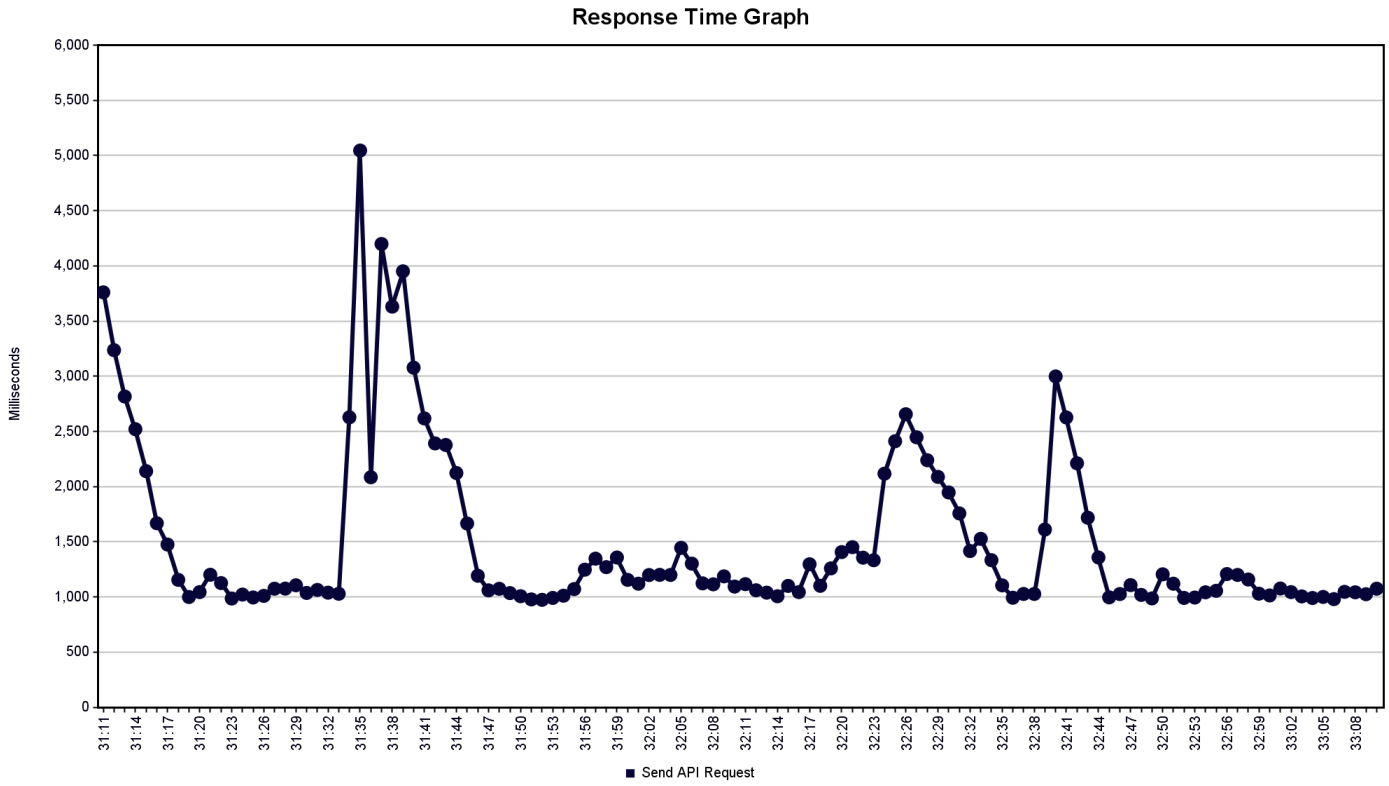


As shown here the test was successfully completed with the required number of requests (10req./sec for 2mins). There were no error and the average response time was 1.5 sec





From the previous 2 screenshots every request was a success with **Credentials verified but waiting on Challenge completion**. (This is due to the Access policy requiring step up authentication)



🔍 Key Observations:

1. Two Mid-Test Spikes (~5,000 ms and ~3,000ms)

- This suggests the server might have been **overloaded**, or suffered from **bottleneck (e.g., CPU/DB bottleneck)**.

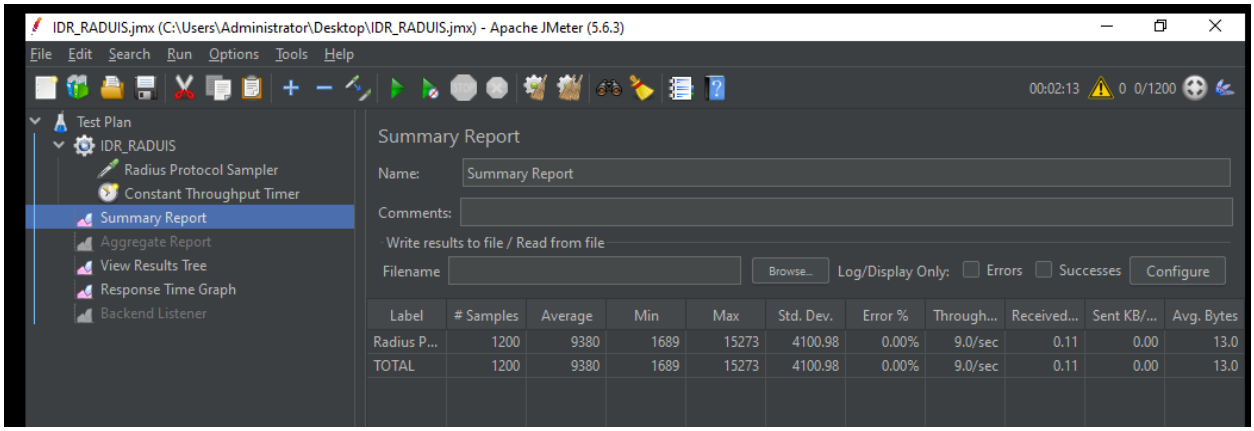
2. Rapid Recovery

- After the spikes, there's a **quick drop** to around ~1,000 ms.
- Indicates that the system recovered after congestion.

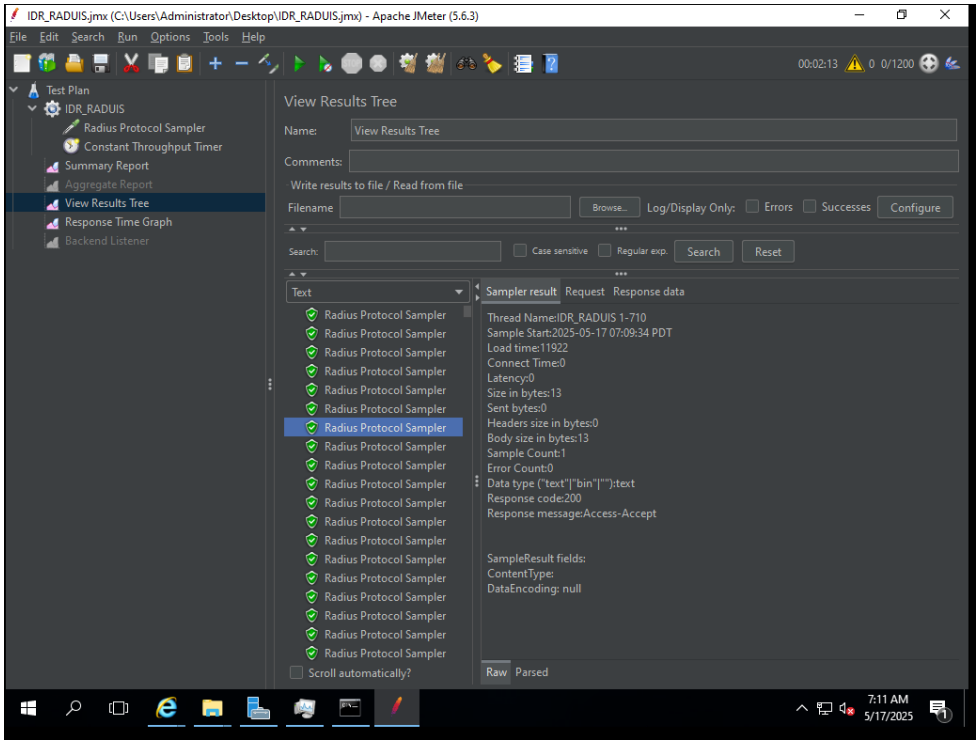
3. Stabilization at ~1,000 ms

- Most response times vary between **1,000–1,200 ms**, suggesting this is the **"normal" baseline latency** under load

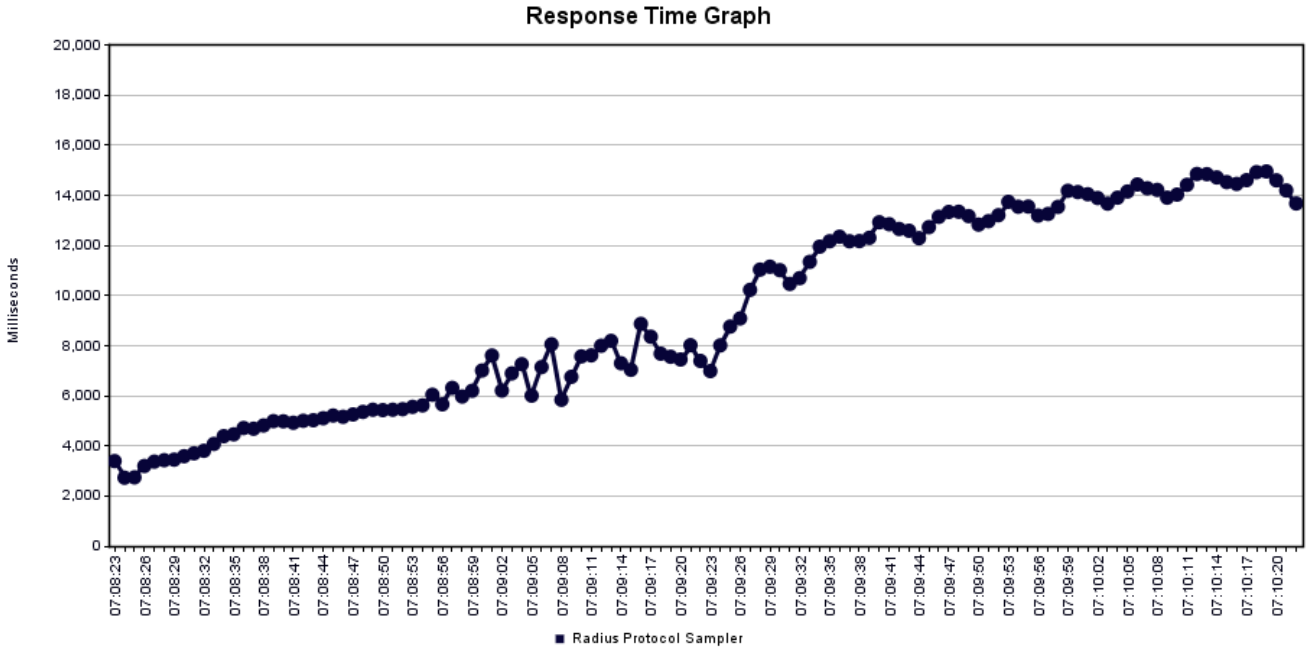
RADIUS_TEST



As shown here the test was successfully completed with the required number of requests (10req./sec for 2mins). There were no error and the average response time was 9.3 sec which was much higher than that of CAS (1.5 sec)



From the previous screenshot every request was a success with a response msg of **Access-Accept**. (This is due to changing the Access policy to allow users to authenticate without step up authentication)



🔍 Key Observations:

1. Steady Increase in Response Time

- The graph shows a **progressive rise** in response time over the course of the test indicating **increasing latency** as load or time progresses.
- Despite the rising response time, the test does **not show a crash or complete failure**, which means the system is **functionally resilient**, though **not optimal**.

2. Possible Bottlenecks or Resource limitation:

- The upward trend suggests that either the **IDR is becoming overloaded**, or there is a **resource limitation** (CPU, memory, or thread pool saturation) as more RADIUS authentication requests are processed. Also, the bottleneck may be due to the capabilities of the VM itself since the IDR is on a VM on a local host not cloud hosted as the CAS.

3. Performance Degradation:

- The performance **does not stabilize**; instead, it **degrades over time**. This could lead to **timeouts or failed authentications** in larger number of requests.

Recommendation:

- Investigate **resource utilization** on the IDR during both types of testing (CPU/memory/thread stats) this can be done by varying the requests/sec as was intended at first.
- Tune **testing configurations** that could better simulate the required **requests/sec.**
- Hosting the IDR on **cloud-based server** could produce better results achieving **less response times**.