

# RiVS: Reputation in VoIP Systems

Bruno Freitas Cruz<sup>a</sup> and Bruno Sousa<sup>b</sup>

*Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal*

**Keywords:** VoIP, SIP, IP PBX, Signalling, Reputation Systems, Internet Communications.

**Abstract:** Voice-over-IP (VoIP) technology revolutionised telecommunications through inexpensive and flexible communications. Its growing user base worldwide across various sectors, including business, healthcare, and telecommunications, originates from the rising demand for VoIP services, which exposes VoIP systems to multiple threats. RiVS combines a reputation system based on the Alpha-Beta distribution to further enhance the security of VoIP systems to protect against toll fraud, authentication attacks, and Spam over Internet Telephony (SPIT). RiVS is integrated with the Asterisk server (known as IP PBX or VoIP server) operating as an intermediary between the VoIP server and the reputation system. RiVS is able to parse user authentication or call attempts to determine if they should be flagged, dropped, or allowed through the system. The reputation system keeps records of positive and negative actions. Experimental procedures were conducted to evaluate system resources usage and response time. With results showing actions performed promptly with few system resources, for consideration of cloud and on-premises deployments.

## 1 INTRODUCTION


Over the years, Voice-over-IP (VoIP) technology's rapid expansion, due to its cost-effectiveness and flexibility, has significantly increased the associated security threats. The convenience and accessibility of VoIP systems made them appealing targets for malicious activities such as Spam over Internet Telephony (SPIT), unauthorised access, and toll fraud. Posing a financial and privacy issue. Several reports (businessWire, 2023; Association, 2023; Insights, 2023; Carter, 2023; gra, 2016) endorse these developments.


Reputation systems are capable of continuously assessing the behaviour of network participants, allowing real-time identification and response to threats, and are capable of incorporating dynamic and adaptive security solutions. Allowing for more targeted and efficient security management and evaluate users based on ongoing behaviour. RiVS analyses the actions performed, the call source, and the duration of the call, among other parameters that are used to establish the reputation score of the reputation system.

To evaluate RiVS performance, the Asterisk VoIP server was chosen, as it is open source and widely used as an IP PBX solution in enterprises. RiVS was integrated with the Asterisk server and with the rep-

utation system to provide real-time monitoring, to report activities and automatically handle activity based on reputation scores of both Internet Protocol (IP) addresses and users. RiVS was evaluated in a Proof-of-Concept (PoC) considering different types of threads such as toll fraud, authentication attacks, and Spam over Internet Telephony (SPIT). **SPIT** can be broadcast to any IP phone or server connected to the Internet, and these calls continue to increase rapidly. The idea is to create a genuine-looking message capable of deceiving protection mechanisms to generate spam calls or attempt to reach as many users as possible to increase the chances of selling a product. These calls can cause serious discomfort for the recipients due to their intrusive nature. The **Authentication attacks** include attempts by bad actors to gain unauthorised access by trying to register using default manufacturer credentials or weak credentials. This is commonly performed via brute-force or dictionary attacks. The **Toll Fraud** refers to generating costs associated with using a hijacked user extension. An attacker uses the IP PBX to make international calls or calls to premium numbers, potentially using it to collect revenue.

The remainder of this paper is organised as follows. Section 3 provides information on the reputation system. Section 4 describes the developed security module, named VoIP WatchDog, which connects

<sup>a</sup>  <https://orcid.org/0009-0000-7873-7203>

<sup>b</sup>  <https://orcid.org/0000-0002-5907-5790>

being the bridge between Asterisk and the Reputation System. Section 5 details the use cases for testing the security module. Section 6 concludes the paper.

## 2 RiVS: REPUTATION IN VoIP SYSTEMS

### 2.1 Reputation System

The reputation system framework is composed of two distinct components, the reputation system itself and RabbitMQ. RabbitMQ is an open-source and asynchronous message broker that processes messages between systems and applications, routing messages according to defined parameters. The reputation system analyses the received data and calculates the score using the Alpha-Beta model. Reputation is calculated by dividing the total number of positive events by the sum of all positive and negative events, which means that the result will vary between 0 and 1. However, multipliers can change the range of these results. This reputation model also contains an ageing factor that is responsible for determining how much past actions can influence new score calculations.

### 2.2 VoIP WatchDog - Security Module

The VoIP WatchDog Module is made up of six Python scripts. The module is portrayed in Figure 1. Two are related to Asterisk authentication, three to Call Routing, and one to parse the Call Records/Reports. The authentication scripts perform different actions. The Call Routing scripts are fed by Asterisk with SIP headers information. Call Tracker and Call Tracker Cleaner complement each other, as without Cleaner, the Tracker would not be viable in a test or production environment. The Call Reports automatically fetch data from Asterisk. This module bridges interactions between Asterisk and the Reputation System, dynamically blocks registration or call attempts, continuously monitors for premium/international calls, and tracks internal calls.

The main building blocks of the VoIP WatchDog module include several components. The **CDR Parser** automatically fetches and processes call records when these are available in the Call Detail Record (CDR) by going through the *uniqueid*, interpreting the database table. The **Security.log Parser** will read and interpret the authentication logs of the *security.log* file and send the information in JSON format to the Reputation System to determine the reputation scores. The **ACL Handler** checks for IPs with

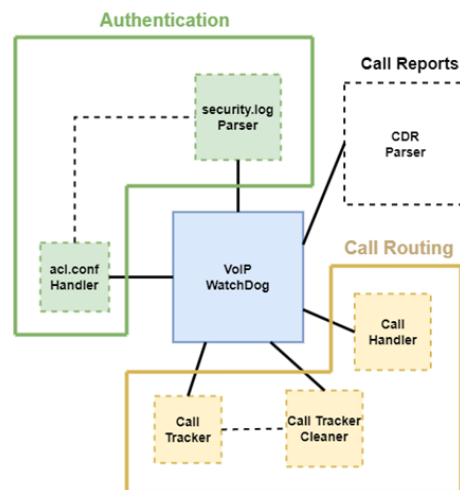


Figure 1: Security Module Framework.

a reputation score below five points in the database of the Reputation System. It also configures the *acl.conf* file to add IPs that meet the aforementioned criteria but remove them if the reputation score exceeds five. The **Call Handler** is fed with information and activated by Asterisk through the Dial Plan, when a SIP INVITE message arrives. It will log into the Reputation System database and check the user score database, so if one of the users has a score lower than five, the call is terminated. The **Call Tracker** is activated and fed by Asterisk through the dial plan, after the SIP Header information is extracted. This component tracks calls from internal users using the active calls database, which is automatically fed when a SIP INVITE message arrives from the internal users, being used to prevent a user on an ongoing call from performing multiple calls. The **Call Tracker Cleaner** is similar to Call Handler and Call Tracker, this component is also fed and triggered by Asterisk A Dial Plan, but after one of the sides terminates the call and receives the SIP BYE message. It ensures that the active calls database is cleaned when an explicit call is terminated.

## 3 EVALUATION METHODOLOGY

### 3.1 Methodology

The experimental scenario consisted in different virtual machines with an Asterisk Server and a SIP (Session Initiation Protocol) trunk, machines to run soft phone software (MicroSIP, Zoiper and Jitsi), a machine with the reputation system and the Rab-

bitMQ instance, and a Kali machine for introducing load or attacks in the remaining components. Three experimental procedures were prepared to evaluate the RiVS framework and system integration: Access Control Lists, Flagging Users, and Call Monitoring.

### 3.2 Access Control Lists

Figure 2 represents the experimented scenario for brute-force emulation, where the security module identifies malicious authentication attempts, flags, and blocks IPs. Two different IPs were used to test the current scenario of the reputation-based access control list (ACL) configuration. The first Internet Protocol (IP), 10.0.0.5, has a score of 2,90 in the IP score database, while 172.0.0.4, belonging to the Kali machine, has a score of 6. The first IP is an entry from preparatory tests, and as a result, it is already present in the *acl.conf* file with a low score.

Using the Kali machine, a word list with random passwords was used to emulate user enumeration, where eight attempts were generated. The moment VoIP WatchDog detects new entries in *security.log*, it parses them and sends the appropriate information in JSON to RabbitMQ, which will queue and route the information to the Reputation System to calculate the reputation score and allow it to be stored; so many attempts caused the score to go lower than five (5). After updating the newly calculated reputation score in the Reputation System database and communicating the updated score via RabbitMQ, the module updates the *acl.conf* file. It reloads the Asterisk ACL module as a new IP with a score less than five (5) can be found. From here on, any authentication from these IPs is no longer allowed.

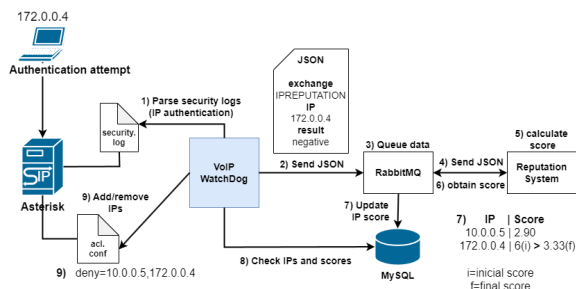


Figure 2: Access Control List Trial.

### 3.3 Flagging Users/Extensions

Figures 3 presents a scenario of identifying users through Call Detail Records (CDR) for the identification of spam. Extension 221 initiates the extension call from behind Asterisk B. Asterisk A receives the call request on the trunk and forwards it to extension

103. At this point, Asterisk already knows that the call is externally sourced. The call lasted more than thirty seconds, so after the call was finished, the CDR was immediately filled with the records of that communication. The VoIP WatchDog immediately parses the new entry in the CDR. The user is flagged as positive, and the module sends a message to RabbitMQ in JSON format. This queues and sends data to the reputation system for reputation score calculation. A new entry is created in the Reputation System database with user scores if the user does not have a previous record. As the first flag for extension 221 is positive, it is assigned a reputation score of 6.

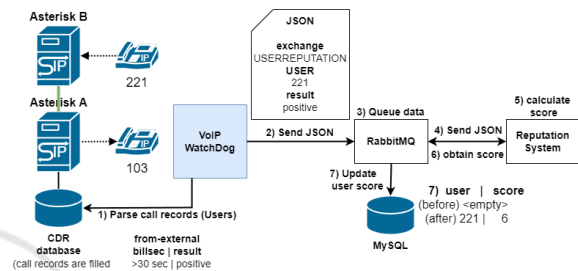


Figure 3: Flagging Users Positively.

Otherwise, if it were a negative flag for the first time, the assigned reputation score would be 4. It should be noted that the reputation score is initiated with the value five (5) by default, which means that there is not enough information to determine whether an entity is trustworthy or not.

### 3.4 Call Monitoring

VoIP WatchDog always monitors the user score and active calls. For the first scenario, the same user was logged in to two different softphone applications. The first call reached extension 221, filling the active call database with the ongoing call information. When the second call was initiated from the other application, it was terminated, thanks to the existing entry for tracking calls; otherwise, it would have gone through. Entries are cleared by Asterisk right after the call has been terminated, leading to the interception of multiple calls from the same source.

Secondly, the call is automatically terminated if any user in a call attempt has a score lower than five (5). Scenario II considers that if both the source and destination have a score equal to or higher than five (5), the calls proceed (From: 102, To: 221); otherwise, they are rejected (From: 598, To:102). This applies in both directions.

Third, it tracks whether a bad actor is maliciously trying to establish calls from a network other than the Asterisk A local network (192.168.1.0/24). This call

is prevented and the source IP signalled with a negative flag. Otherwise, without this procedure, the call will be registered as sourced from Asterisk A, resulting in costly fees. So, even if the trunk credentials get exposed for unknown reasons, or there is an unknown exploit that would allow generating a call externally, this provides an additional security layer. Allows rejection and flagging of calls from unknown IPs that could self-identify as internal users, as shown by Figure 4.

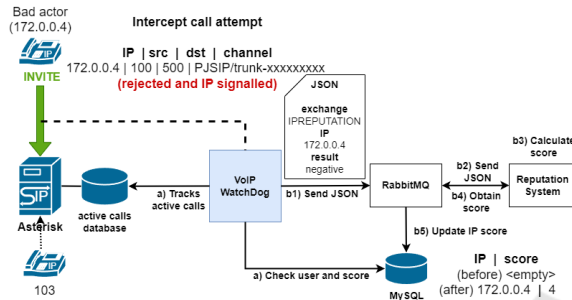


Figure 4: Call monitor - Scenario III.

## 4 EVALUATION RESULTS

### 4.1 Access Control Lists

The time achieved for Step 1 - Parse security logs and Step 2 - Send information to RabbitMQ is between 0.001 and 0.005 milliseconds (ms) to parse each row individually. The time discrepancy is related to the amount of content parsed when searching for the relevant conditions.

The results achieved for Step 4 - Send information to the reputation system, Step 5 - Calculate the reputation score, and Step 6 - Propagate the reputation score took on average 2 ms to complete the message request for the RabbitMQ score calculation and send it back to update the IP score database.

Step 7, which refers to updating the score database in RabbitMQ docker, is not available. However, following previous observations and tests, it should not take more than a few milliseconds. This characteristic is shared by all scenarios. Figure 5 refers to steps 8 and 9 of figure 2.

- In red are the IPs fetched from the IP score database in RabbitMQ docker. Connecting to the database and fetching all IPs and respective scores took 6.397 ms. In another attempt, the same action took only 2.805 ms.
- In orange is how long the module took to configure the *acl.conf* file with the newly founded IP, taking only 0.159 ms.

- In yellow is the time to reload the Asterisk ACL module after reconfiguring the ACL file, taking only 5,812 ms. It takes 5,971 ms to configure *acl.conf* and reload the Asterisk ACL module for the new configuration to take effect.
- In blue is the CPU utilisation. The CPU usage was always recorded as less than 1% remaining at 0.00% the whole time. With cycle processing time representing the entire cycle time, from data extraction from the database to configuring *acl.conf* and reloading the Asterisk ACL module for the configuration to take effect.
- In green is the RAM usage. Under normal operations, the ACL component only uses 23.43 Mb and, at most, would make use of 31.79 Mb.

Disk read / write is at 0.00 Mb; however, writing is at 0.01 Mb, which should correspond to editing the *acl.conf* file. The network utilisation is shown at 0.00 Mb.

```

2024-06-29 20:18:45,221 - Fetched IP: 10.0.0.5, Score: 2.9869767441860463
2024-06-29 20:18:45,221 - Fetched IP: 127.0.0.1, Score: 6.0
2024-06-29 20:18:45,221 - Fetched IP: 127.0.1.1, Score: 6.0
2024-06-29 20:18:45,221 - Fetched IP: 127.0.2.1, Score: 6.0
2024-06-29 20:18:45,221 - Fetched IP: 127.127.0.1, Score: 6.0
2024-06-29 20:18:45,221 - Fetched IP: 169.254.104.20, Score: 7.093023255813954
2024-06-29 20:18:45,221 - Fetched IP: 172.0.0.3, Score: 6.03157894736842
2024-06-29 20:18:45,221 - Fetched IP: 172.0.0.4, Score: 3.3333333333333334
2024-06-29 20:18:45,221 - Fetched IP: 192.168.0.1, Score: 6.363636363636363
2024-06-29 20:18:45,221 - Fetched IP: 192.168.1.5, Score: 7.707263389581804
2024-06-29 20:18:45,221 - Fetched IP: 199.99.99.99, Score: 5.079363887572246
2024-06-29 20:18:45,222 - Database IP fetch time: 0.006397 seconds
2024-06-29 20:18:45,222 - IPs with score < 5: ['10.0.0.5', '172.0.0.4']
2024-06-29 20:18:45,222 - Added IPs: ('172.0.0.4')
2024-06-29 20:18:45,222 - Removed IPs: set()
2024-06-29 20:18:45,222 - ACL configuration updated successfully
2024-06-29 20:18:45,222 - ACL configuration update time: 0.000159 seconds
2024-06-29 20:18:45,228 - Asterisk ACL module reloaded successfully
2024-06-29 20:18:45,228 - ACL reload time: 0.005812 seconds
2024-06-29 20:18:45,228 - Total ACL configuration and reload time: 0.005971 seconds
2024-06-29 20:18:45,228 - Cycle processing time: 0.012604 seconds
2024-06-29 20:18:45,228 - Total CPU time for cycle: 0.000000 seconds
2024-06-29 20:18:45,228 - CPU usage for cycle: 0.00% over 0.012604 seconds with 2 CPUs
2024-06-29 20:18:45,228 - Memory usage: RSS=23.43 MB, VMS=31.79 MB
2024-06-29 20:18:45,228 - CPU usage: 0.00%
2024-06-29 20:18:46,230 - Memory usage: RSS=23.43 MB, VMS=31.79 MB
2024-06-29 20:18:46,230 - Disk I/O: Read=0.00 MB, Write=0.01 MB
2024-06-29 20:18:46,230 - Network I/O: Sent=0.00 MB, Received=0.00 MB
    
```

Figure 5: ACL benchmarking.

Ten samples were retrieved to calculate how quickly IPs are blocked from initial detection to score calculation and *acl.conf* file configuration. Furthermore, it always considers updating the file with any IPs that should be added, removed, or kept, depending on the database scores.

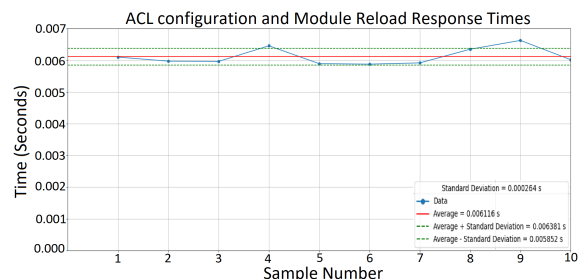


Figure 6: ACL response times.

According to figure 6, it took an average of 6.116 ms, with a variation of 0.0264 ms. The longest time was 6.381 ms and the lowest was 5.5852 ms. Such values indicate an acceptable performance for ACL functionalities in the RiVS framework.

### 4.2 Flagging Users/Extensions

This section documents the performance and utilisation of the resources of the figures 3 Steps 1 and 2.

The CPU utilisation varies. The processing of 1605 lines took 260 ms with a CPU utilisation of 40.45%. This includes the parsing and sending in JSON format to RabbitMQ. The processing of a new line, leads to CPU utilisation at 0.00%. The RAM utilisation, under normal operations, the ACL component only utilises 27.57 Mb and, at most, would make use of 107.64 Mb. The Disk I/O read is at 0.00 Mb, however, writing is at 0.36 Mb, this value should correspond to the module logging the parsing and sending of each line of the CDR database. And network utilisation is shown with a value of 1.17 Mb after sending 1605 JSON messages.

The operation of fetching a new CDR record has the following resource utilisation. The RAM utilisation increased by 0.02 Mb, but since it is the same module and running continuously, it could be related to a momentary memory speak. The Disk I/O write operation increases to 0.37 Mb. This is because the module logging will print data related to fetching the CDR every 5 seconds.

The performance of performing Step 1 - Parsing call records depicted in Figure 3 relies on the step 1 average and deviation response times for parsing 1716 rows in the Call Detail Record (CDR) database. An average response time of 270 ms was achieved, which translates to 0.1573 ms to process each row. The samples always matched the same processing time, without deviation, that is, the standard variation is 0.000000 ms.

The steps Step 4 - Send information to reputation system, Step 5 - Determine reputation score, Step 6 - Communicate reputation score the achieved values to determine the reputation score rely on the order of 2 ms, which is inline with the value achieved in the access control list scenario.

Figure 7 presents the average and deviation response times for the reputation system score calculations, which lead to an average of 2.1 ms. The horizontal red line indicates an average response time of 2.1 ms. For most requests, the response times reside close to the average line, indicating consistency. The standard variation is 0.567646 ms. Even if most samples reside close to the average processing time, there

are a few outliers. This could be related to a few events that occasionally impact response times within the system.

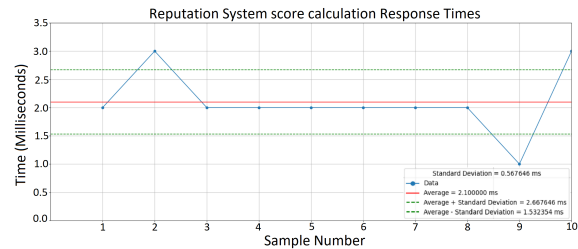


Figure 7: Reputation System score calculation response times.

The performance values achieved were in line with the results of the access control list scenario. This shows stability in the RiVS framework in assessing the reputation score for VoIP information.

### 4.3 Call Monitoring

The utilisation of resources of scenario I with respect to active call tracking includes the time to query data from the database, the CPU and RAM utilisation rates. Connecting to the active call database took 3.209 ms. And querying for any active call source from extension 103 took only 0.0384 ms. The time it took to fill the database with the ongoing call is 1.335 ms. However, it can be obtained by subtracting the database query and the database connection time from the overall processing time. The measured CPU rate is on the order of 0.00%, while the RAM usage under normal operations, the call tracking component only uses 26.60 Mb and, at most, would use 106.84 Mb.

Figure 8 provides the performance and use of resources of Scenario II. The utilisation of resources of scenario II includes the same metrics as in scenario I. The connection to the database takes only 0.059 milliseconds. The time to fetch the extension score is 0.83 milliseconds. The hangup action includes logging into AMI (Asterisk Management Interface) and executing the hangup command, taking only 0.181 ms. The CPU utilisation is 0.00% and the RAM utilisation under normal operations, the hangup component only utilises 23.49 Mb and, at most, would use 31.86 Mb.

Figure 9 provides the results of the use of resources in scenario III provides the performance and use of the resources in Figure 4. The connection to the database to check active calls took 3.248 ms, and querying for active calls took 0.305 ms. The detection of incoming call IP trying to reach extension 221 and block the malicious attempt and sending a JSON

```

2024-06-30 17:09:41,325 - Getting connection from pool
2024-06-30 17:09:41,325 - Connection pool acquisition time: 0.000059 seconds
2024-06-30 17:09:41,325 - Checking score for from_extension: 598
2024-06-30 17:09:41,326 - Query time for from_extension: 0.000588 seconds
2024-06-30 17:09:41,326 - from_extension: 598, from_score: 4.0
2024-06-30 17:09:41,326 - Checking score for to_extension: 102
2024-06-30 17:09:41,326 - Query time for to_extension: 0.000242 seconds
2024-06-30 17:09:41,326 - to_extension: 102, to_score: 6.666666666666666
2024-06-30 17:09:41,326 - Hanging up call on channel: PJSIP/trunk-00000001
2024-06-30 17:09:41,327 - AMI hangup time: 0.000181 seconds
2024-06-30 17:09:41,328 - CPU times start: user=0.0400, system=0.0100
2024-06-30 17:09:41,328 - CPU times end: user=0.0400, system=0.0100
2024-06-30 17:09:41,328 - Number of CPUs detected: 2
2024-06-30 17:09:41,328 - Overall processing time: 0.003041 seconds
2024-06-30 17:09:41,328 - Total CPU time: 0.000000 seconds
2024-06-30 17:09:41,328 - CPU usage: 0.00% over 0.003041 seconds with 2 CPUs
2024-06-30 17:09:41,328 - Memory usage: RSS=23.49 MB, VMS=31.86 MB
2024-06-30 17:09:41,328 - Disk usage: Read=0.00 MB, Write=0.00 MB
    
```

Figure 8: Hangup Scenario II benchmarking.

```

2024-06-30 18:18:33,359 - Database connection acquisition time: 0.003248 seconds
2024-06-30 18:18:33,359 - Database query execution time: 0.000305 seconds
2024-06-30 18:18:33,359 - Active calls for trunk: ()
2024-06-30 18:18:33,359 - Hanging up call on channel: PJSIP/trunk-00000000 due to disallowed I
P: 172.0.0.4 for extension: 221
2024-06-30 18:18:33,360 - Time taken to hang up call on channel PJSIP/trunk-00000000: 0.000123
seconds
2024-06-30 18:18:33,367 - Sent ('IP': '172.0.0.4', 'result': 'negative') to RabbitMQ
2024-06-30 18:18:33,370 - Overall processing time: 0.014074 seconds
2024-06-30 18:18:33,370 - Total CPU time: 0.000000 seconds
2024-06-30 18:18:33,370 - CPU usage: 0.00% over 0.014074 seconds with 2 CPUs
2024-06-30 18:18:33,370 - Memory usage: RSS=26.71 MB, VMS=178.84 MB
2024-06-30 18:18:33,370 - Disk usage: Read=0.00 MB, Write=0.07 MB
2024-06-30 18:18:33,370 - Network I/O: Sent=0.01 MB, Received=0.01 MB
    
```

Figure 9: Hangup Scenario III benchmarking.

message took 10.4 ms, which correlates with a TLS connection. The CPU usage is 0.00%, while using RAM under normal operations, this part of the hangup component only uses 26.71 Mb and, at most, would use 178.84 Mb. The network data display both 0.01 Mb, which can be correlated to receiving calls and sending the call termination packets.

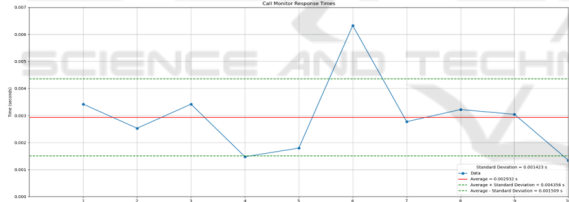


Figure 10: Call Monitor response times.

Figure 10 presents the average and deviation response times for the Call Routing component of VoIP WatchDog. The blue dots represent the sample number. The X-axis is the sample number and the Y-axis is the response time in ms. The horizontal red line represents the average. The horizontal red line indicates an average response time of 2.932 ms. Most of the time, response times reside close to the average line, indicating consistency. The standard variation is 1.423231 ms. Response times spread across a relatively significant area, suggesting considerable variation. Most response times reside within the variation area, especially in the lower parts, and some are close to the average.

The average response time is higher in this scenario, as it includes the processing from the WatchDog component in the RiVS framework, compared to

the previous scenarios. Notwithstanding, the achieved results (i.e., 2.932ms) are still acceptable for VoIP communications.

#### 4.4 RabbitMQ Messaging

In order to test RabbitMQ capabilities, the CDR sent 3242 lines of information in individual JSON messages. At its peak, RabbitMQ successfully processed more than 800 messages per second without dropping any of them, as shown in Figure 11. RabbitMQ took 20 seconds to process all 6484 messages. This means that processing of each message takes 3.083 ms in average.

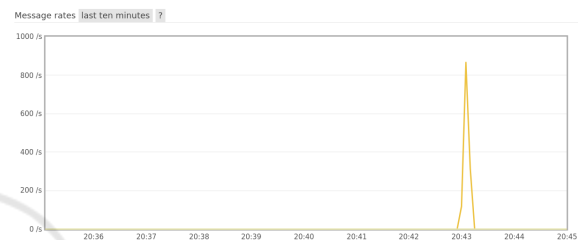


Figure 11: RabbitMQ - Messages processing rate.

The choice of RabbitMQ demonstrates that it is possible to have a reliable communication channel between the components of the RiVS framework.

#### 4.5 Docker Container's

The resource utilisation figures presented here were taken during the previous section scenario exercise of processing 3242 data lines from the CDR. Considering message processing capabilities, the number of lines took around four (4) seconds to receive and process through the Reputation System. Any additional processing time is related to post-JSON data. In the upcoming figures, a dot represents the start, and a line at the end of the 3242 lines that were processed. Past that point, it is related to post-JSON actions, which involve reputation update routing policies. In the figures, red is the CPU utilisation, followed by the RAM in orange and network utilisation in green.

Reaching a spike of 0.4 in the Cores axis of RabbitMQ indicates that 20% of the processing power was utilised during an average time frame of around 4 seconds (Asterisk A is running with 2 cores). With figure 13 running stable slightly above 0, on standby, and innately increasing when data needs to be processed.

RabbitMQ in figure 15 and the Reputation System in figure 16 showed RAM (Random Access Memory) efficiency. But the same cannot be said about MySQL database in figure 17.

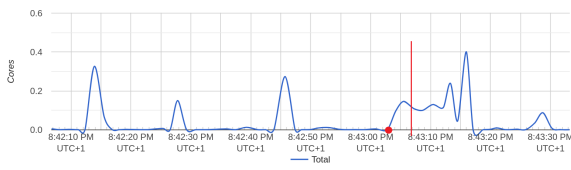


Figure 12: RabbitMQ CPU.

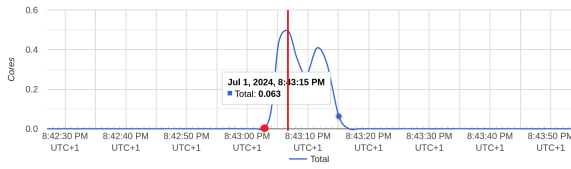


Figure 13: Reputation System CPU.

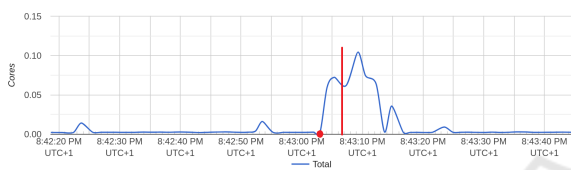


Figure 14: MySQL CPU.

When queuing and processing over 3242 JSON packets, RabbitMQ used 175 megabytes (Mb) of memory, spiking slightly above 200 megabytes when routing traffic for the Reputation System and between score updates in between the MySQL database and the Reputation System.

In these Memory figures, *Total* represents all the memory allocated to the container, regardless of whether it is utilised. At the same time, *Hot* refers to the amount of memory actively used.

After initialisation, the Reputation System was actively using 60 Mb of RAM, from a maximum of 100 Mb if required. After processing the 3242 packets, it started actively using 125 Mb and around 170 Mb and remained stable throughout further experiments.

The MySQL database in figure 17 was using slightly more than 475 Mb and was increasing close to 487,5 Mb but capable of reaching more than 500 Mb in use. The reason for so much RAM is related to not only containing asterisk records but also information for RabbitMQ and the Reputation System.

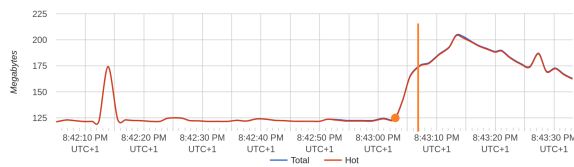


Figure 15: RabbitMQ Memory.

The network utilisation is very low and will not easily cause network congestion. In Figure 18, 3242 JavaScript Object Notation (JSON) messages

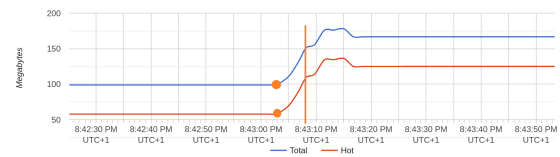


Figure 16: Reputation System Memory.

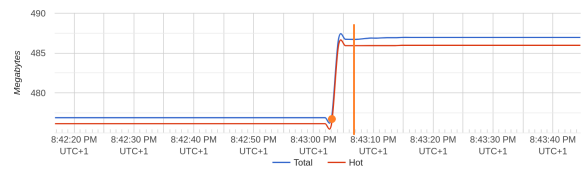


Figure 17: MySQL Memory.

generated incoming network traffic of up to 0.45 Megabyte(Mb)/s = 450 000 Bytes per second.

Subsequently, received (Rx) bytes are related to reputation update exchanges, with the same type of traffic applied to transferred (Tx) bytes, including the 3-way handshakes. Reputation exchanges are separated into two parts: First comes the score calculation and then updating the score.

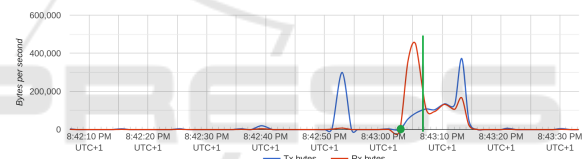


Figure 18: RabbitMQ Network usage.

In figure 19, Tx reputation score was 0.25 Mb/s. However, the Rx information from the last updated scores led to a slight increase above 0.3 Mb/s. In figure 20 a Tx rate of 0.05 Mb/s and Rx rate of 0.01 Mb/s peak around 0.15 Mb/s.

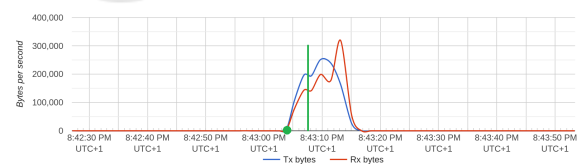


Figure 19: Reputation System Network usage.

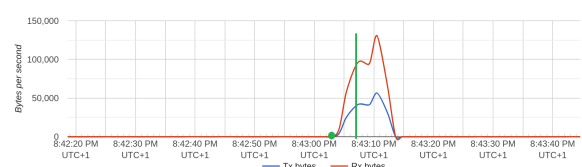


Figure 20: MySQL Network usage.

After sending the calculated reputation update, RabbitMQ will provide the Reputation System with

the latest calculation. When a packet arrives in the Reputation System, it opens it, performs the calculations, and replies to RabbitMQ. Hence, RabbitMQ job is to queue these *reputationUpdate* messages and forward them to the MySQL and Reputation System.

So, the remaining bulk of the network traffic past the green line is the traffic associated with what is observed in figure 21 and is applied to figures 19 and 20. Figure 21 shows an example of a calculated score for extension 200 but then came in another JSON with a positive flag to increase the extension score. Hence, the Reputation System does its job by again calculating the score and releasing a new update for which RabbitMQ will route the *reputationScore* messages to inform the database for VoIP WatchDog access and the Reputation System.

```
RECEIVED MESSAGE from [reputationUpdates : #]: {"currentScore":4.99999999997726,"SRC":"200"}
Processing time: 0ms
RECEIVED MESSAGE from [reputationUpdates : #]: {"currentScore":4.99999999998863,"SRC":"200"}
Processing time: 0ms
```

Figure 21: Reputation Update from RabbitMQ.

## 5 CONCLUSION AND FUTURE WORK

Voice-over-IP (VoIP) will continue to rise due to the need to have continuous and inexpensive communication methods. However, this continuously brings about various security challenges, such as Spam over Internet Telephony (SPIT), toll fraud, and brute-force attacks, endangering the integrity and dependability of VoIP networks. In response to these threats, the RiVS framework explored the implementation of reputation systems as a proactive security measure, offering a way to improve the resilience of VoIP systems against malicious actors.

Although the CDR parser component effectively addresses the identification of SPIT according to the defined set of parameters, the integration of Machine Learning (ML) would present a substantial refinement. The parameters involved defining a value on the call's duration to determine whether a call was spam.

During the evaluation, the performance results show that actions are performed quickly and the system remains stable. For example, on average, to process an individual event, RabbitMQ takes 3.083 milliseconds (ms) to process each message, Reputation System 2.1ms for score update, CDR Parser 0.1573ms for processing a row, Call Routing Module 2.932ms to handle calls, and Authentication Module 6.1744ms to configure and restart the Access Control List (ACL) for immediate effect. Some variations can be observed, but are not unreasonable according to the output. Demonstrating efficiency and stability, which

can contribute to future enhancements.

However, VoIP WatchDog was built using Python scripts, meaning that a new shell process runs every time it is executed, consuming resources every time, even if by a small amount. Due to limitations, performance bottlenecks are not noticeable in such an experimental PoCs, and the volume of calls is low compared to the volume generated in a production environment, where they could have serious bottlenecks.

As future work, we consider the following: 1) Enhance spam detection through the implementation of ML. 2) Implement an Asterisk Gateway Interface (AGI), to take advantage of FastAGI, to improve all operations involving database connections, log parsing, call control and reduce resource consumption further and the load on the Asterisk Server, since the security module requires to be installed on the Server. 3) Deploy the Reputation System in the cloud, to emulate a public facing service.

## ACKNOWLEDGEMENTS

This work implemented a reputation system that was developed through the ARCADIAN-IoT project, sponsored by the European Union's Horizon 2020 research and innovation programme and supported under grant agreement number 101020259. This work has been supported by Project "Agenda Mobilizadora Sines Nexus". ref. No. 7113), supported by the Recovery and Resilience Plan (PRR) and by the European Funds Next Generation EU, following Notice No. 02/C05-i01/2022, Component 5 - Capitalisation and Business Innovation - Mobilising Agendas for Business Innovation. This work is also funded by National funds through the FCT- Foundation for Science and Technology, I.P., within the scope of the project CISUC-UID/CEC/00326/2020 and by the European Social Fund, through the Regional Operational Program Centro 2020.

## REFERENCES

(2016). Mobile voip market size to reach \$145.76 billion by 2024. <https://www.grandviewresearch.com/press-release/global-mobile-voip-market>. Accessed: 2023-11-06.

Association, C. F. C. (2023). Telecommunications fraud increased 12% in 2023 equating to an estimated 38.95 billion lost to fraud. <https://bit.ly/4byliLF>. Accessed: 2024-01-03.

businessWire (2023). Hiya report: A quarter of all unknown calls in the world are spam and fraud. <https://www>.



tmcnet.com/submit/2023/05/02/9805850.htm. Accessed: 2024-01-02.

Carter, R. (2023). The ultimate list of voip statistics (2024). <https://findstack.com/resources/voip-statistics/>. Accessed: 2023-11-06.

Insights, G. M. (2023). Voice over internet protocol (voip) market. <https://bit.ly/3xNZAXp>. Accessed: 2023-11-03.

