Cybersecurity 2025/2026

Alessandro Modelli
0001145565

Project Report

Summary

Introduction

This report describes the design, implementation and analysis of a controlled ransomware simulation within a cloud infrastructure on Google Cloud Platform (GCP).

The primary goal of this project was to reproduce the typical behavior of a ransomware script during execution, with a particular focus on the encryption and exfiltration of the victim's data to an attacker's server, and to evaluate the effectiveness of a detection system based on network analysis, capable of activating automated response measures.

The simulation environment was designed according to the principles of isolation, control, and observability, using a Virtual Private Cloud (VPC) structured with two subnets to differentiate management and simulation activities, firewall rules, network flow logging enabled via VPC Flow Logs, and subsequent export to a BigQuery dataset for analysis.

The simulation is divided into multiple phases:

- Configuration of the infrastructure on Google Cloud Platform
- Configuration of the ransomware execution scenario
- Execution of the simulation
- Monitoring and analysis of the collected logs
- Design and implementation of a detection system
- Design and implementation of automated containment measures
- Testing of the automated detection system and analysis of the collected flows

The outcome allows the observation of the effects of automated containment measures, activated by a detection system, on a running ransomware script.

Phase 1: Infrastructure configuration on Google Cloud Platform (GCP)

1.1 Virtual Private Cloud (VPC)

In order to execute the simulation within an isolated and controlled environment, a Virtual Private Cloud named rs_lab_vpc was created and divided into two subnets according to the following architecture:

VPC Architecture

Subnet	VM	Description	IP Address
management-subnet		Access/Control	10.10.10.0/24
	bastion-vm		10.10.10.3
simulation-subnet		Simulation	10.10.20.0/24
	attacker-vm	Attacker	10.10.20.3
	victim-vm	Victim	10.10.20.2
	Client-1	Client	10.10.20.4
	Client-2	Client	10.10.20.5

The VPC was divided into the following two subnets with the purpose of separating the simulation instances from the control instance:

- management-subnet (1 VM – Bastion):
    - Region: europe-west8 (Milan)
    - IPv4 Range: 10.10.10.0/24
- simulation-subnet (4 VMs – Attacker, Victim, Client1, Client2):
    - Region: europe-west8 (Milan)
    - IPv4 Range: 10.10.20.0/24

To enable network traffic collection, VPC Flow Logs must be activated on the simulation-subnet.

1.2 Virtual Machines

As anticipated in the previous section, five GCP virtual machine instances were used for this project, each with a specific role. The virtual machines were created with configurations specifically selected to ensure efficient resource utilization:

VM	**Name**	**Type**
Bastion	bastion-vm	E2 Small (2 vCPU, 1 core, 2GB memory)
Victim	victim-vm	E2 Small (2 vCPU, 1 core, 2GB memory)
Attacker	attacker-vm	E2 Small (2 vCPU, 1 core, 2GB memory)
Client-1	client1-vm	E2 Micro (2 vCPU, 1 core, 1GB memory)
Client-2	client2-vm	E2 Micro (2 vCPU, 1 core, 1GB memory)

All instances use Debian 12 (Bookworm) as the operating system, x86_64 architecture, in order to ensure cost efficiency, stability, and security with respect to the simulation.

For an isolated and controlled environment, external IP addresses were removed from the client1-vm and client2-vm instances, preventing direct communication with the Internet. External IP connectivity was retained for the attacker-vm and victim-vm instances to allow installation of the libraries required by the Python scripts used during the simulation.

1.3 Firewall

To maintain strict access control within the VPC, a set of restrictive firewall rules was implemented to prevent direct access to the simulation instances, allowing SSH connections only through the bastion-vm.

- rs-lab-deny-all-ssh
    - Blocks any SSH connection attempt from any IP address to any destination.
    - Priority: 1001
    - Destination: 0.0.0.0/0
    - Source: 0.0.0.0/0
    - Action: Deny TCP:22
- rs-lab-allow-ssh-to-bastion
    - Allows SSH connections from any source to the bastion-vm; this way, the bastion-vm acts as the single entry point to the simulation environment.

    - Priority: 1000
    - Destination: bastion-vm
    - Source: 0.0.0.0/0
    - Action: Allow TCP:22
- rs-lab-allow-ssh-bastion
    - Allows SSH traffic from the management-subnet to the simulation-subnet; this enables connections from the bastion-vm to the simulation instances.
    - Priority: 1000
    - Destination: 10.10.20.0/24
    - Source: 10.10.10.0/24
    - Action: Allow TCP:22
- rs-lab-vpc-allow-custom
    - Allows all types of traffic between the two VPC subnets.
    - Priority: 65534
    - Destination: 0.0.0.0/0
    - Source: 10.10.10.0/24, 10.10.20.0/24
    - Action: Allow all traffic

This firewall configuration ensures that access to the simulation environment is tightly controlled while still allowing necessary internal communication between VPC subnets.

1.4 Access and management

Access to the virtual machine instances was designed in a centralized manner, based on the use of a bastion host. In this configuration, access to the simulation instances (victim-vm, attacker-vm, client1-vm, client2-vm) is performed exclusively via SSH connections from the bastion-vm, which is accessible via SSH from the GCP console or through the interface button. This approach ensures that the environment remains isolated.

Once SSH keys have been configured on the various instances, it is sufficient to execute the following command from the bastion-vm's console:

ssh [vm tag]

1.5 VPC Flow Logs configuration

Once the VPC Flow Logs option was enabled on the simulation subnet, it was necessary to create a VPC Flow Logs configuration named gcp-fl-simulation-subnet-europe-west on the simulation-subnet in order to start collecting network flows, with the following settings:

- Log aggregation interval of 5 seconds
- Inclusion of all metadata

1.6 BigQuery configuration

Once log collection was configured, automatic log export was set up to ensure an efficient analysis. A new dataset named rs_logs was created in GCP BigQuery, and log export was implemented by creating a sink that automatically forwards all logs collected by the VPC Flow Logs configuration to the selected dataset.

BigQuery enables SQL-based querying of the data, facilitating data analysis and the detection of anomalies or suspicious activities such as data exfiltration.

1.7 Phase 1 Testing

Test 1 – Internal Connectivity

Verification of communication between virtual machines inside the simulation subnet via ping.

Test 2 – Access Control

Verification of the ability to establish SSH connections from the bastion-vm to the simulation virtual machines.

Test 3 – External Isolation

Verification that simulation virtual machines do not respond to connection attempts originating from outside the VPC.

Test 4 – VPC Flow Logs Collection

Verification that the VPC Flow Logs configuration correctly collects and exports logs related to network traffic on the simulation subnet.

Phase 2: Configuration of the ransomware execution scenario

2.1 HTTP requests

For a realistic network traffic scenario, the python script *httpGen.py*[1] was implemented on each virtual machine instance client1-vm, client2-vm and victim-vm to generate bidirectional client–server traffic.

The httpGen.py script is designed to generate HTTP requests toward a server specified via the url parameter (client1-vm targets the server running on client2-vm and vice versa). The execution duration of the script is configurable through the duration parameter, which is set to 60 seconds by default. It is also possible to configure the waiting time between requests using the interval parameter, which is set to 4 seconds by default. The actual waiting interval is then calculated randomly between 80% and 120% of the specified value. This approach ensures that the generated network traffic appears more realistic and casual.

2.2 Target files

To simulate the effects of ransomware execution, a directory named *files* containing 250 dummy files was created on the victim-vm instance. These files serve as test data and were generated as targets for the encryption and exfiltration operations performed by the ransomware script. The procedure for their creation is outlined below.

[1] https://github.com/alessandromodelli/ransomware-simulation-with-detection-and-containment/blob/main/httpGen.py

All scripts used during the simulation on victim-vm are stored within the *scripts* directory.
After creating the *files* directory, the *fakeFilesGen.py*[2] script was executed with the *files* directory specified as the target path and a file generation count of 250:

python3 scripts/fakeFilesGen.py --target files --amount *250*

Executing this command resulted in the generation of 250 test files with different extensions, including .txt, .csv, .pdf, .jpg, and .png, and with variable sizes from 256 bytes to 10 MB.
The total volume of generated data is equal to 746 MB.

2.3 Exfiltration server

In order to simulate data exfiltration activity from the victim-vm, the Python script *server.py*[3] implementing an HTTP server was implemented on the attacker-vm. This component represents the attacker-controlled server within the simulated scenario and is responsible for receiving files transferred from the victim machine.

To prepare the execution environment, the required packages for python dependency management and virtual environments were installed:

sudo apt install -y python3-pip python3-venv

Right after that, a dedicated Python virtual environment was created and activated for running the server.py script, ensuring isolation of application dependencies:

python3 -m venv venv
source venv/bin/activate

Within the virtual environment, the Flask library was installed and used to implement the HTTP server:

pip install flask

The server.py script exposes a /status endpoint listening on 10.10.20.3:8080, intended to receive the files exfiltrated from the victim-vm. Received files are stored locally in the uploads/victim-vm directory, simulating the successful completion of the data exfiltration process.

2.4 Ransomware script

Next, the ransomware script *ransomware.py*[4] was implemented directly on the victim-vm to simulate the local files encryption and exfiltration to the attacker's server.
Just like for the attacker-vm, a python virtual environment was configured.

First of all, the required packages were installed:

---

[2] https://github.com/alessandromodelli/ransomware-simulation-with-detection-and-containment/blob/main/fakeFilesGen.py
[3] https://github.com/alessandromodelli/ransomware-simulation-with-detection-and-containment/blob/main/server.py
[4] https://github.com/alessandromodelli/ransomware-simulation-with-detection-and-containment/blob/main/ransomware.py

sudo apt install -y python3-pip python3-venv

After thant, the python virtual environment was created and activated:

python3 -m venv venv
source venv/bin/activate

Within the virtual environment, the Cryptography and Requests libraries were installed. These libraries are used respectively for handling file encryption operations and for sending HTTP requests to the attacker's server:

pip install cryptography requests

What the script *ransomware.py* does is exfiltrating each file inside the target directory (passed as parameter) to the attacker's server, creating a secret key used for the encryption and encrypting each file.

2.6 Phase 2 testing

To ensure the correct setup prior to the execution and analysis phases, the following validation tests were performed:

Test 1 – Network traffic

Verification of the generation of HTTP traffic between the client virtual machines, confirming the correct execution of the traffic simulation scripts.

Test 2 – Target directory and files

Verification of the presence of the target directory on the victim-vm and confirmation of the existence of the predefined number of dummy files created for the ransomware simulation.

Test 3 – Exfiltration server script

Verification that the server.py script is present on the attacker-vm, ensuring the availability of the attacker-side infrastructure.

Test 4 – Ransomware script verification

Verification that the ransomware.py script is present on the victim-vm and ready for execution.

Phase 3: Simulation

After completing the simulation setup, the actual simulation was executed.

3.1 VPC Flow Logs collection

The first step of the simulation phase consisted of enabling the VPC Flow Logs configuration on the simulation subnet in order to collect network traffic logs generated during the execution of the scenario.

From GCP, the configuration is activated by navigating through the menu to VPC, then VPC Flow Logs, selecting Subnet under project-level configurations, and enabling the configuration named

gcp-fl-simulation-subnet-europe-west8. Once activated, the subnet begins generating flow logs according to the predefined aggregation interval and metadata settings, ensuring that all network communications occurring during the simulation are properly captured for subsequent analysis.

3.2 HTTP requests generation

At this stage, an HTTP server listening on port 8080 was started on both client1-vm and client2-vm. The servers were executed using nohup to ensure that they continued running in the background even after the SSH session is terminated.

On client1-vm:

nohup python3 -m http.server 8080 > http.log 2>&1 &

Su client2-vm:

nohup python3 -m http.server 8080 > http.log 2>&1 &

For each instance, the server logs are written to a file named http.log in the directory from which the command is executed. The server execution can be stopped at any time by terminating the corresponding process.

Once the servers are running, the HTTP request generation script is executed to simulate bidirectional client-server traffic.

From client2-vm to client1-vm:

nohup python3 -u httpGen/httpGen.py --url http://10.10.20.4:8080 --interval 10.0 --duration 1200 > http.log 2>&1 &

From client1-vm to client2-vm

nohup python3 -u httpGen/httpGen.py --url http://10.10.20.5:8080 --interval 10.0 --duration 1200 > http.log 2>&1 &

From victim-vm to client2-vm

nohup python3 -u scripts/httpGen.py --url http://10.10.20.5:8080 --interval 10.0 --duration 1200 > http.log 2>&1 &

The execution of the above commands results in the generation of HTTP traffic over a 20-minute duration, with an average interval of 10 seconds between requests. This traffic represents standard background network activity as a baseline for comparison with the malicious traffic generated during the ransomware execution.

3.3 Exfiltration server execution

At this point, the attacker-side server was started on the attacker-vm in order to receive the files exfiltrated from the victim-vm. The server listens on port 8080.

Before launching the server, the python virtual environment previously configured on the attacker-vm is activated:

source venv/bin/activate

The server is then executed in background:

nohup python3 scripts/server.py > http.log 2>&1 &

All server's logs generated during execution are collected in the http.log file, which can be used to verify the correct reception of exfiltrated files.

3.4 Ransomware execution

Before executing the ransomware, to guarantee that the same set of files were used across multiple simulation runs, the original directory containing the generated files was duplicated and renamed to serve as the ransomware target directory:

cp -r files/ target

At this point, the target directory contains the files used for the simulation, and the ransomware script can be executed. The python virtual environment is first activated and then the ransomware script is launched with the target directory specified as a parameter:

source venv/bin/activate
python3 scripts/ransomware.py --target target

Upon execution, the ransomware starts encrypting all files within the target directory and exfiltrates them to the attacker-vm through the previously configured server. For potential future improvements of the simulation, the secret encryption key used during the file encryption process is also exfiltrated to the attacker-vm.

The next step consisted of analyzing the collected logs in order to identify anomalies and behavioral patterns generated by the execution of the ransomware.

Phase 4: Analysis and Detection

4.1 Log analysis

The logs collected during the simulation were automatically exported, through the dedicated log sink, to the BigQuery dataset rs_logs.

Each VPC Flow Log entry contains multiple fields describing the characteristics of the observed network flow. For the purpose of this analysis, the following properties were selected as the most relevant for detection and anomaly identification:

Proporty	Description
connection	Identifier representing a tuple composed of source IP, source port, destination IP, destination port, and protocol.

reporter	Indicates the perspective from which the flow was observed, either from the source or the destination of the connection.
start _time	Timestamp marking the beginning of the log aggregation interval.
end_time	Timestamp marking the end of the log aggregation interval.
bytes_sent	Total number of bytes transmitted from the source to the destination during the recorded interval.

To simplify the analysis of the collected logs, a flattened view was created to normalize the most relevant fields extracted from the VPC Flow Logs. The following query was used in BigQuery:

```
CREATE OR REPLACE VIEW gruppo1-russo.rs_logs.rs_logs_flat AS
SELECT
  timestamp,
  PARSE_TIMESTAMP('%Y-%m-%dT%H:%M:%E*S%Ez', jsonPayload.start_time) AS start_time,
  PARSE_TIMESTAMP('%Y-%m-%dT%H:%M:%E*S%Ez', jsonPayload.end_time) AS end_time,
  jsonPayload.connection.src_ip AS src_ip,
  jsonPayload.connection.dest_ip AS dest_ip,
  jsonPayload.connection.src_port AS src_port,
  jsonPayload.connection.dest_port AS dest_port,
  CAST(jsonPayload.bytes_sent AS INT64) AS bytes_sent,
  CAST(jsonPayload.packets_sent AS INT64) AS packets_sent,
  jsonPayload.reporter AS reporter,
  jsonPayload.src_instance.vm_name AS src_vm,
  jsonPayload.dest_instance.vm_name AS dest_vm,
  jsonPayload.connection.protocol AS protocol
FROM `gruppo1-russo.rs_logs.networkmanagement_googleapis_com_vpc_flows_20260120`;
```

The first simulation was executed on January 20th at 12:41, generating a total of 1409 log entries. An initial analysis revealed the presence of IP addresses not associated with any of the virtual machines involved in the simulation.

To identify these unknown IP addresses, the following query was executed:

```
SELECT DISTINCT
  CASE
    WHEN src_vm IS NULL THEN src_ip
    WHEN dest_vm IS NULL THEN dest_ip
  END AS unknown_ip
FROM `gruppo1-russo.rs_logs.rs_logs_flat`
WHERE src_vm IS NULL OR dest_vm IS NULL
ORDER BY unknown_ip;
```

The results showed IP addresses belonging to the following ranges, which were subsequently identified using the *whois* command:

- 142.x.x.x -> Google
- 192.178.x.x -> Google
- 216.x.x.x -> Google

After identifying these addresses, all logs related to such external IP ranges were filtered out to obtain a clearer dataset focused exclusively on the simulation environment.

A new view, rs_logs_filtered, was therefore created:

```
CREATE OR REPLACE VIEW `gruppo1-russo.rs_logs.rs_logs_filtered` AS
SELECT *
FROM `gruppo1-russo.rs_logs.rs_logs_flat`
WHERE
    NOT (STARTS_WITH(src_ip, '142.')
    OR STARTS_WITH(src_ip, '216.')
    OR STARTS_WITH(src_ip, '192.')
    OR STARTS_WITH(dest_ip, '142.')
    OR STARTS_WITH(dest_ip, '216.')
    OR STARTS_WITH(dest_ip, '192.'));
```

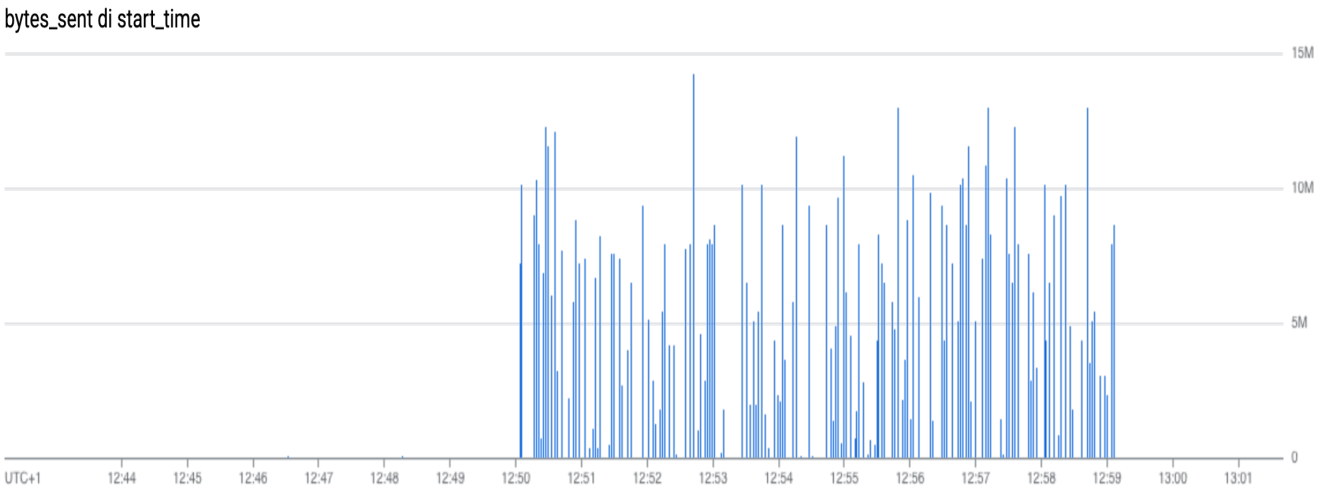
This filtering step reduced the number of log entries from 1409 to 704, significantly simplifying subsequent analysis.

4.2 Traffic analysis and anomaly identification

Using the filtered view, it was possible to distinguish normal network traffic from traffic generated during the ransomware simulation. The resulting graph clearly highlights two main behaviors:

- Ransomware execution phase: a pronounced traffic spike between 12:50 and 12:59, corresponding to the file exfiltration activity toward the attacker virtual machine.
- Baseline network traffic: standard traffic remains minimal, with a maximum observed value of 39.7 KB for bytes_sent, which is negligible when compared to the megabyte-scale traffic generated during the ransomware activity.

This distinction provides a clear foundation for defining detection thresholds and identifying anomalous behavior within the network.



The distribution of destination ports used for communication between the simulation virtual machines was also examined, both under normal conditions and during the ransomware execution phase.

To identify baseline behavior, the following query was executed, explicitly excluding the time window corresponding to the ransomware activity:

```
SELECT
  dest_port,
  COUNT(*) AS logs,
  SUM(bytes_sent) AS total_bytes
FROM gruppo1-russo.rs_logs.rs_logs_filtered
WHERE start_time NOT BETWEEN
  TIMESTAMP '2026-01-20 11:49:12.053190 UTC'
  AND
  TIMESTAMP '2026-01-20 12:00:12.053190 UTC'
GROUP BY dest_port
ORDER BY logs DESC;
```

Under standard traffic conditions, the highest amount of bytes_sent was observed on port 40120, with a volume slightly below 50 KB, while on the port 8080 the volume was under 20KB.

The same analysis was then repeated without excluding the ransomware execution window:

```
SELECT
  dest_port,
  COUNT(*) AS logs,
  SUM(bytes_sent) AS total_bytes
FROM gruppo1-russo.rs_logs.rs_logs_filtered
GROUP BY dest_port
ORDER BY logs DESC;
```

The results show a complete different pattern. During the ransomware execution, the vast majority of network activity concentrates on port 8080, which records:

- 418 log entries
- 1,519,117,024 bytes transmitted

This behavior clearly corresponds to the file exfiltration process toward the attacker server.

4.3 Detection system

As previously observed, under normal conditions the maximum number of bytes_sent on port 8080 remains below 20 KB. For this reason, 25 KB per flow was selected as a reliable detection threshold.

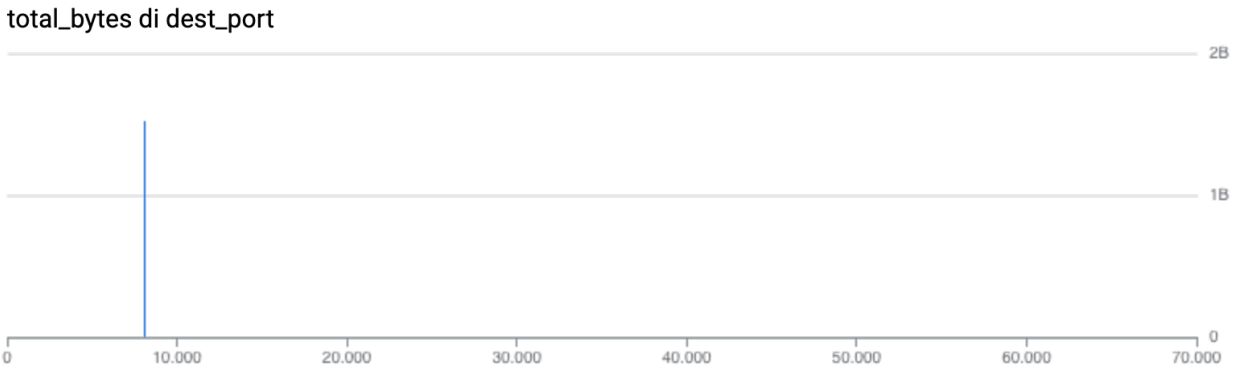
The detection system was designed and implemented based on the following parameters:

- Number of events within a fixed time window
- Minimum bytes sent per event

For this simulation, the detection thresholds were defined as follows:

- Time window: 1 minute
- Minimum number of events: 5
- Minimum bytes sent per event: 50 KB

When the query identifies a number of events exceeding these thresholds, the response measures must be triggered.



The corresponding detection query is shown below:

```
DECLARE WINDOW_MINUTES INT64 DEFAULT 1;
DECLARE MIN_EVENTS INT64 DEFAULT 5;
DECLARE MIN_BYTES_PER_EVENT INT64 DEFAULT 25 * 1024;  -- 25 KB

WITH filtered_logs AS (
  -- Step 1: remove low bytes_sent
  SELECT
    src_vm,
    dest_vm,
    dest_port,
    timestamp,
```

```
    bytes_sent
  FROM gruppo1-russo.rs_logs.rs_logs_filtered
  WHERE
    bytes_sent IS NOT NULL
    AND bytes_sent >= MIN_BYTES_PER_EVENT
),

windowed_logs AS (
  -- Step 2: assign logs to fixed 1-minute windows
  SELECT
    src_vm,
    dest_vm,
    dest_port,
    TIMESTAMP_SECONDS(
      DIV(UNIX_SECONDS(timestamp), WINDOW_MINUTES * 60)
      * WINDOW_MINUTES * 60
    ) AS time_window,
    bytes_sent
  FROM filtered_logs
)

-- Step 3: detection rule with count of events
SELECT
  src_vm,
  dest_vm,
  dest_port,
  time_window,
  COUNT(*) AS event_count,
FROM windowed_logs
GROUP BY src_vm, dest_vm, dest_port, time_window
HAVING
  COUNT(*) >= MIN_EVENTS
ORDER BY time_window DESC;
```

This query returns exactly the set of 1-minute time windows in which multiple high-volume flows were detected. In practice, the results precisely match the ransomware execution interval, ranging from 12:50:23 to 13:02:03.

In other words, the detection logic successfully isolates the time periods during which the ransomware was actively exfiltrating data, providing a robust foundation for automated response and containment.

4.4 Containment measures

Once the collected logs had been analyzed and the ransomware detection logic was defined, it was possible to design and implement the containment measures to interrupt the execution of the ransomware. The containment strategy is based on two main actions:

- Firewall-level isolation
- Shutdown of the suspicious virtual machine

To enable an automated response, a GCP Cloud Function was developed and integrated with a Cloud Scheduler job, configured to run every 2 minutes. The choice of using GCP Cloud Functions along with Cloud Scheduler Job was purely determined by the fact that the overall infrastructure was implemented in GCP.

The implemented Cloud Function, named *firewall_automation*, executes a detection query at each invocation. The query scans network flow logs from the last 4 minutes, grouping them into 1-minute time windows, and evaluates whether anomalous traffic patterns are present.

Specifically, the function checks for:

- A number of events greater than a predefined threshold (5 events)
- Each event exceeding a minimum size threshold of 25 KB of bytes sent
- Repeated communication toward the same destination

If these conditions are met, the log is classified as potential data exfiltration.
When an anomalous behavior is detected, the following containment measures are automatically enforced:

- Firewall rule creation
    - A new firewall rule is dynamically generated to deny all EGRESS (outbound) TCP traffic from the identified source VM to any destination (0.0.0.0/0).
      This rule is scoped specifically to the compromised VM using network tags, ensuring minimal impact on the rest of the infrastructure. It could be changed to deny all traffic for a total isolation.
- Virtual machine shutdown
    - After the firewall rule is applied, the suspicious virtual machine is automatically stopped, immediately terminating the execution of the ransomware and preventing further encryption or exfiltration activity.

The python script *main.py*[5], of the Cloud Function, inside the Cloud Function folder of GitHub, implements the automated detection and containment logic described above. The function integrates BigQuery for log analysis and Compute Engine APIs for firewall management and VM control:

Lo step successivo è stato eseguire una seconda simulazione con le stesse configurazioni della precedente, ma con il sistema di detection e contenimento automatizzato in funzione.

By combining an almost real-time log analysis with automated containment actions, the system is able to:

- Detect ransomware data exfiltration within minutes
- Immediately isolate the compromised VM on the network
- Fully stop any script execution through automated VM shutdown

[5] https://github.com/alessandromodelli/ransomware-simulation-with-detection-and-containment/blob/main/Cloud%20Functions/main.py

4.5 Automated detection system in action

The next step of the project consisted of running a second simulation using the same configuration as the first one, this time with the automated detection and containment system enabled, in order to evaluate its effectiveness in mitigating the ransomware attack in real time.

Within a time window comparable to the one of the first simulation (approximately 20 minutes), the differences between the two scenarios are immediately evident.

In the initial simulation, data exfiltration was first observed at 2026-01-20 11:50:03.191630 UTC and terminated at 2026-01-20 11:59:06.145904 UTC, resulting in a total duration of 542.954274 seconds (9 minutes and 3 seconds).

In the second simulation, where automated detection and containment mechanisms were enabled, data exfiltration was first detected at 2026-01-21 16:10:37.902954 UTC and was interrupted at 2026-01-21 16:14:24.218936 UTC, with a total duration of 226.315982 seconds (3 minutes and 46 seconds).

This significant reduction in exfiltration time is a direct consequence of the detection system successfully isolating the suspicious virtual machine. As a result, the exfiltration process was interrupted at approximately 40% of the total, with 98 files out of 250 successfully transmitted before containment measures were activated.

These results demonstrate the effectiveness of the automated detection and response pipeline in limiting both the duration and the overall impact of the ransomware-driven data exfiltration.

Final results

This project focused on building and testing a realistic ransomware simulation inside a Google Cloud Platform environment, from the initial setup to the final detection and response system. The work started with a controlled and isolated virtual infrastructure and then moved step by step through the simulation of the ransomware attack, the collection of network logs, their analysis, and the implementation of automated mechanisms to detect and stop the attack.

The simulation clearly showed that the ransomware behavior, with the focus on file encryption and data exfiltration, leaves clear traces in the network level. By collecting the VPC Flow Logs and analyzing them in BigQuery, it was possible to understand what “normal” network traffic looks like, identify suspicious pattern, and define detection thresholds based on how often events occur and how much data is being transferred.

The design and implementation of an automated detection and response system made a significant difference on the ransomware execution. With a scheduled network traffic logs analysis and dynamically generated firewall rules, along with instances control, the system was able to react quickly to suspicious activity.

Compared to the initial simulation without automation, the second run showed a much shorter exfiltration phase, resulting in the compromised virtual machine being successfully isolated and powered off in a short period.

In conclusion, this project confirms that analyzing network logs is an effective and practical approach for detecting and responding to ransomware security incidents.

Resources

https://github.com/alessandromodelli/ransomware-simulation-with-detection-and-containment