UML LAB 3 – MODELING A FORTISANDBOX SYSTEM

1 OBJECTIVE

The objective of this lab is to model the software for a **FortiSandbox**. **FortiSandbox** is a system that analyzes potentially malicious files transiting on a network. The system uses virtual machines (VMs) running different operating systems to test a file and to determine if it is malicious. The **FortiSandbox** system interacts with **FortiGate** as shown in the diagram below.

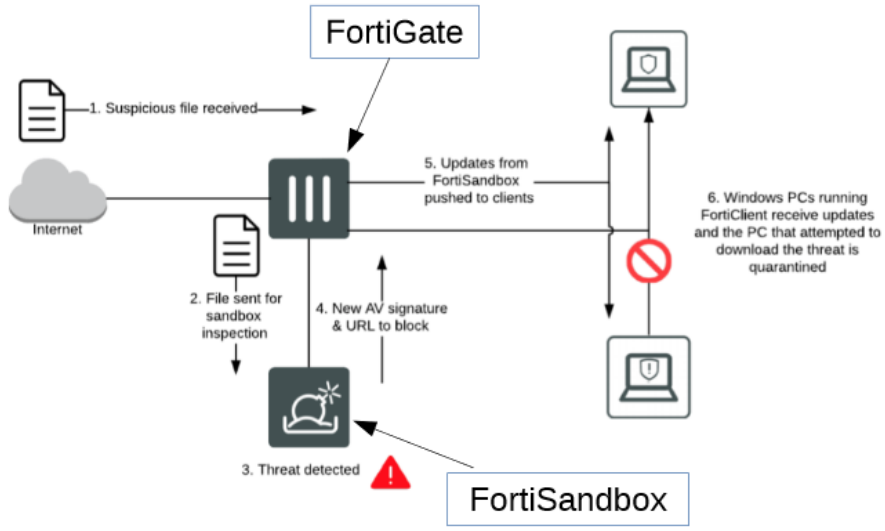

Figure 1- FortiSandbox system environment

The specifications define the following requirements :

- **FortiSandbox** has a VM pool and processes multiple files simultaneously.
- It can take 60 seconds to five minutes to process a file.
- When a **FortiGate** learns from **FortiSandbox** that an endpoint is infected, the administrator can quarantine the host, if it is registered to a FortiClient.

In this report we first present our assumptions for the system. Then we define the requirements using a *requirement diagram* and present an analysis using a *use-case* diagram. Finally, we explain our design and the design choices, using simulation reports to verify the properties of the application.

2 ASSUMPTIONS

A complete summary of our assumptions is found in the diagram below.

Figure 2- Assumptions for the FortiSandobx system model

In our modeling of the system we decided to represent the complete environment of **FortiSandbox**, including the **FortiGate** and the clients. The files are represented as signals coming from the internet, they are assigned to a client randomly. Since the **FortiSandbox** environment can analyze files for multiple users, we decided to have two clients in the model. These clients can receive files from the internet or an alert from the **FortiSandbox** environment if the files are malicious.

To simplify the design and for clarity, our **FortiSandbox** system will use a pool of two VMs. Considering that one VM can analyze one file, our sandbox can effectively analyze two files simultaneously. If another file is submitted while both VMs are busy, the file is ignored and won't be analyzed.

If a file is malicious, our **FortiSandbox** can ban the url from which the file comes from. In our system, we implemented a single url ban, which means that only one url can be banned at a time.

Finally, we assume that the VMs testing the files never fail to detect a virus. Therefore, our system does not have to handle the case where a report gives a false positive report (a report where a malicious file successfully passed the security tests).

3 REQUIREMENTS

Considering our assumptions, the requirement diagram is as follows.

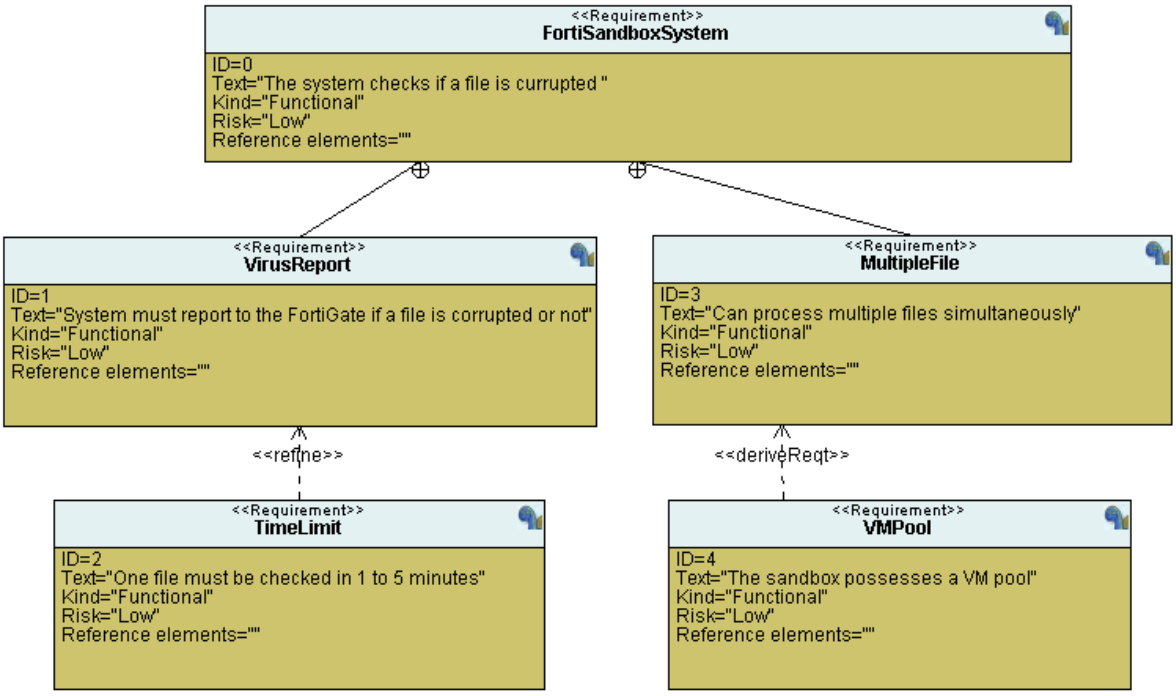

Figure 3- Requirement diagram

4 ANALYSIS

We analyze the system with a use-case diagram.

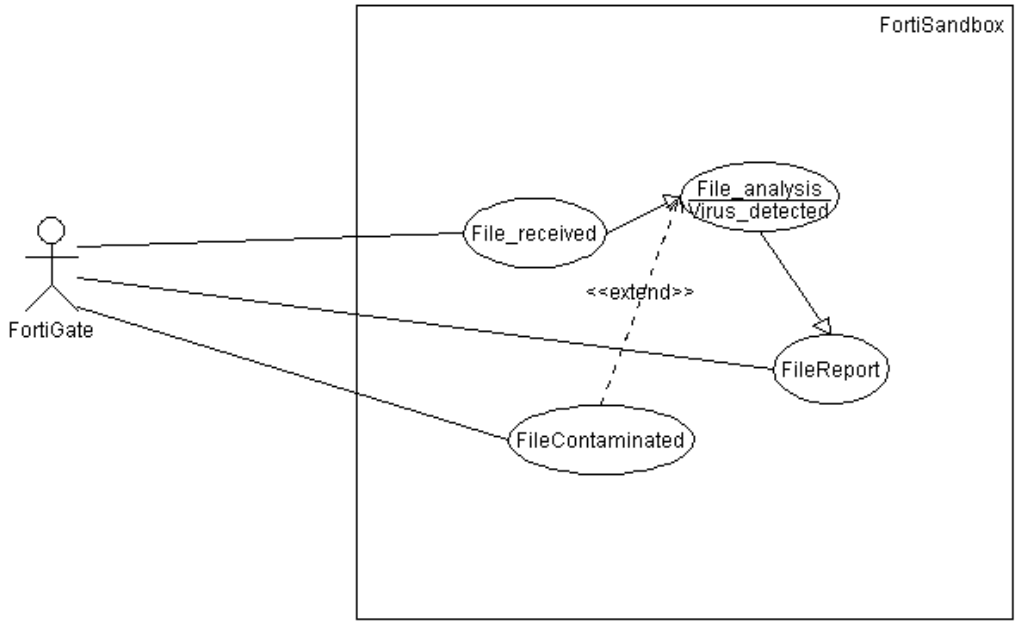

Figure 4- Use Case diagram

In the system environment, the **FortiSandbox** only interacts with one other part, the **FortiGate** system. This is made clear in the use-case diagram, the **FortiGate** is the only actor from the point of view of the **FortiSandbox**.

A file is always received and analyzed when the **FortiSandbox** is needed, and it always returns a file report. The "FileContaminated" activity is only required when a file is considered malicious, thus it is an "extend" activity.

5 DESIGN

For the block diagram, we can take inspiration from the environment schematic provided by the specifications and work things from here.

To simulate the incoming files, we add an "Internet" box sending the files after a given time. Each file has a random boolean value representing if the file is malicious or not, a random client id (either 1 or 2) representing the user that requested the file, and a random url represented by a number between 1 and 3.

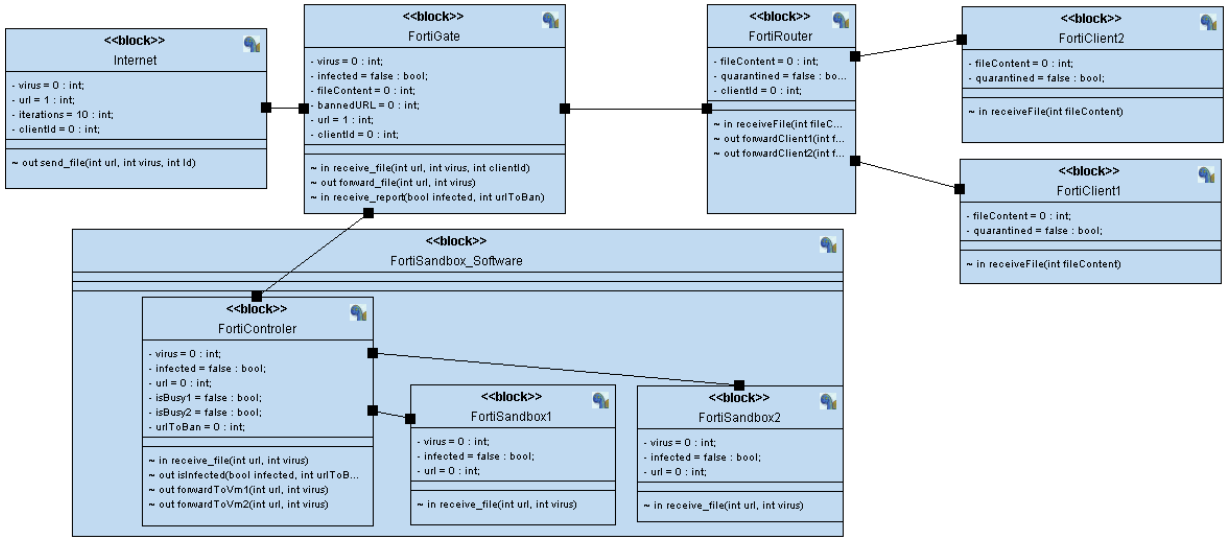

Figure 5- Block diagram

The FortiSandBox_Software box contains the **FortiSandbox** system. It is composed of two VMs named "FortiSandboxN" and a controller that manages the two VMs to select which file to send on which VM.

The **FortiGate** block "intercepts" the file from the internet and checks if the url is banned or not. If it is banned, then the file is ignored. Otherwise it is sent to the **FortiSandBox** for testing. The **FortiGate** receives reports from the **FortiSandbox** and then decides to send the file to the users or not.

The **FortiClients** represent the end users receiving the files and/or an alert message to indicate a malicious file. The **Forti router** dispatch the files and alerts to the correct clients.

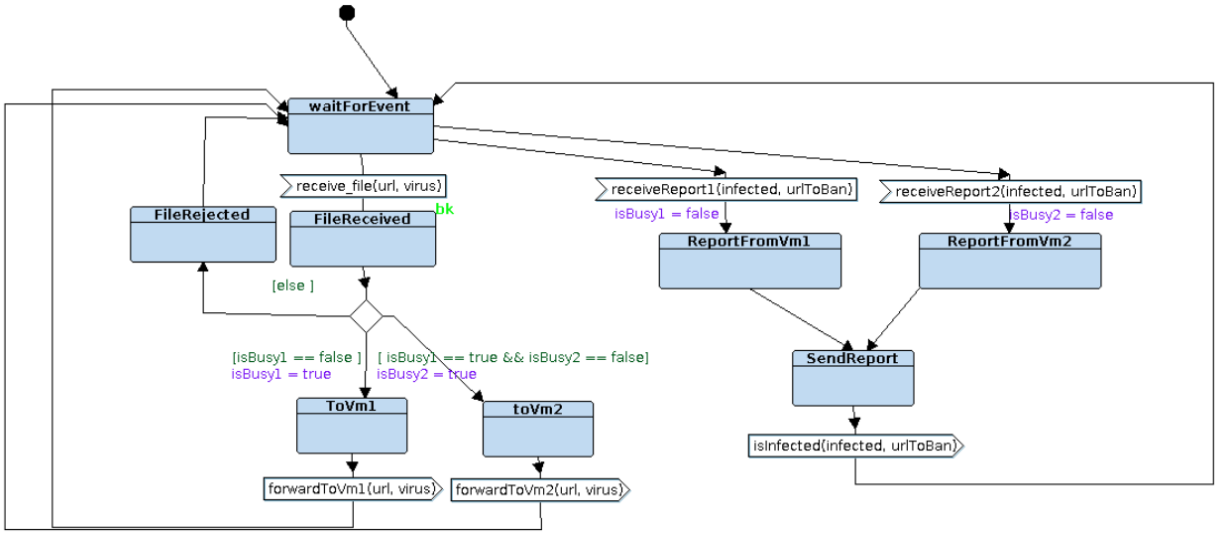

Figure 6- FortiController

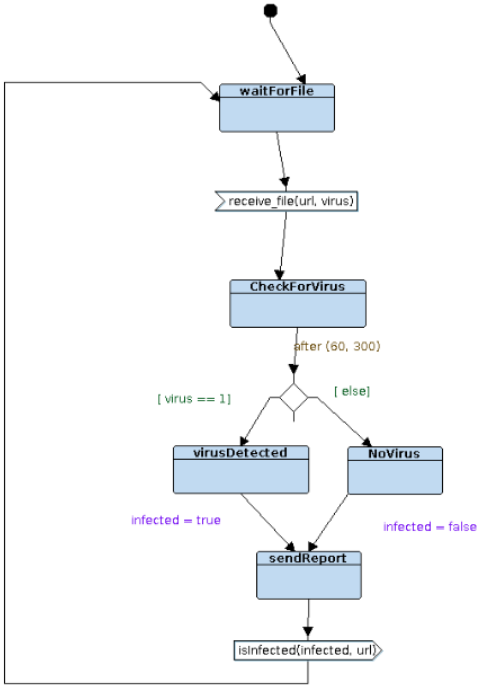

Figure 7- A FortiSandbox

The **FortiController** manages two **FortiSandboxes**. It stores the current state of each VM in variables "isBusyX" and updates these variable whenever a file is received, or a report is ready to be sent. When receiving a file, the controller determines which VM the file must be sent to.

- If VM1 is available, send the file to VM1
- Else if VM is available, send the file to VM2
- Else reject the file.

The sandboxes take between 60 and 300 seconds to complete.

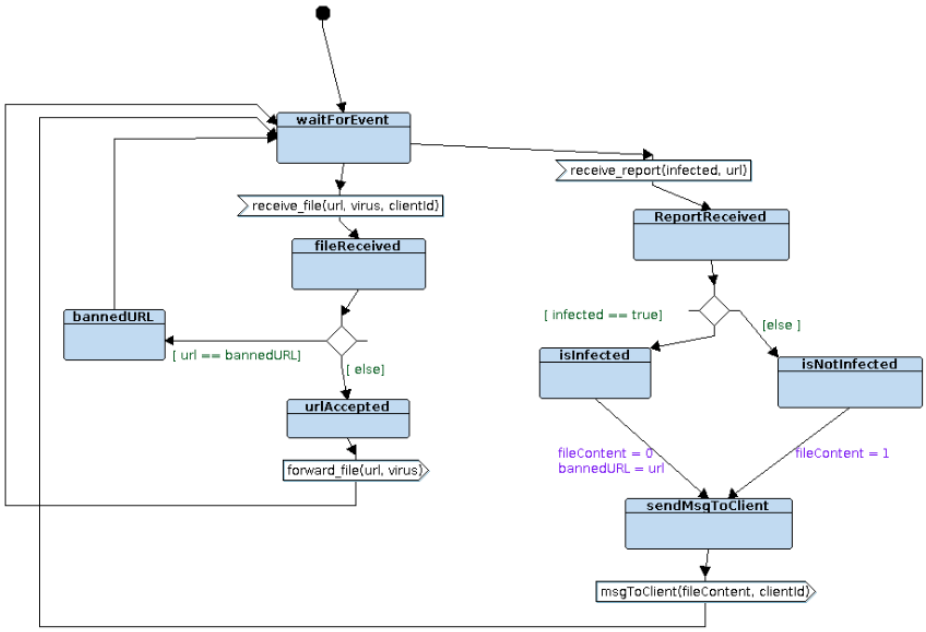

Figure 8- FortiGate

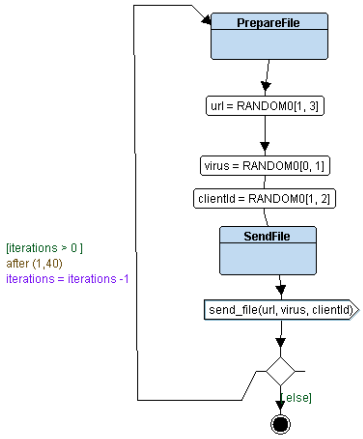

Figure 9- Internet

The **FortiGate** waits for a file or a report to arrive. When a report is received, if it indicates a malicious file then save the current url as the banned url and send an alert message to the client (symbolized by "fileContent = 0"), otherwise send the full content of the file (symbolized by "fileContent = 1").

The internet testbench generates a file with random url, clientId, and virus value. In our simulation we generated 10 files.

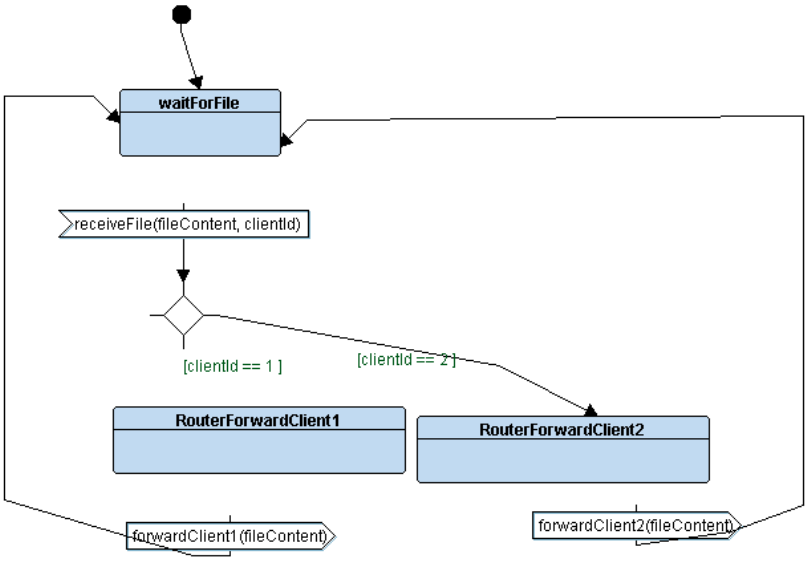

Figure 10- FortiRouter

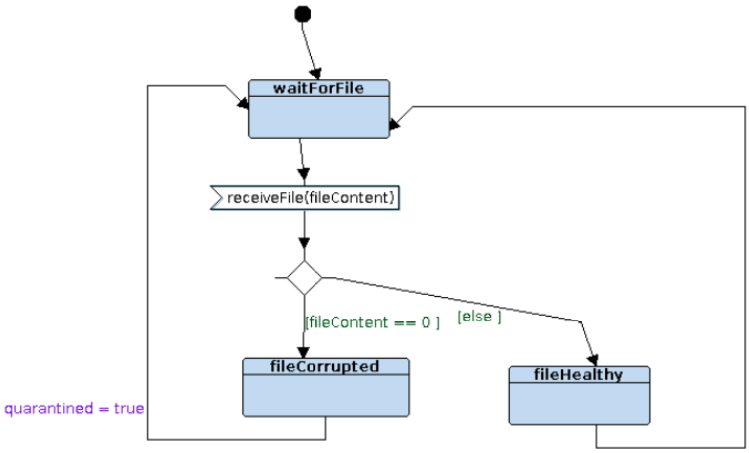

Figure 11- A FortiClient

A **FortiClient** waits for the file to arrive. If an alert message is received (symbolized by "fileContent = 0"), then it is quarantined. The value of the "quarantined" variable currently does not alter the system function, but it symbolizes that the user has requested a malicious file, and it can be used as a base for adding quarantine features to the model.

When simulating with TTool, we get the following traces. When a file is received, we identify three situations :

- VM1 is available and receives the file to analyze (regardless of the state of VM2)
- VM1 is busy but VM2 is available. VM2 receives the file to analyze
- Both VMs are busy and the file is rejected

Also, we check that a url that sent a malicious file is banned, and that subsequent files from that url are rejected without being analyzed.

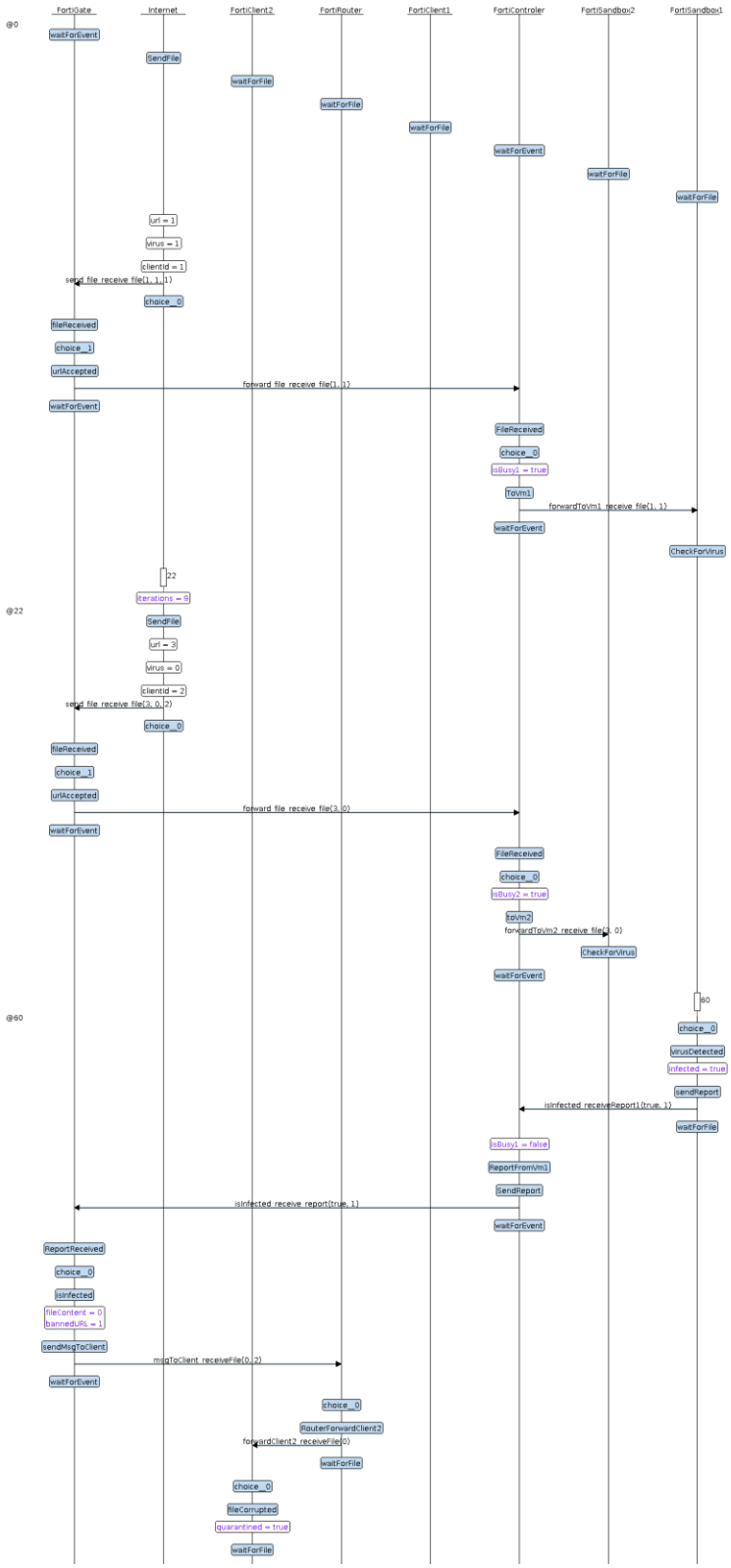

Figure 12- Files sent to VM1 then VM2

This trace shows the first and second situations. Both VMs are available at the beginning of the simulation. The first received file is sent to VM1 (at time 0), then the next file is sent to VM2 (at time 22).

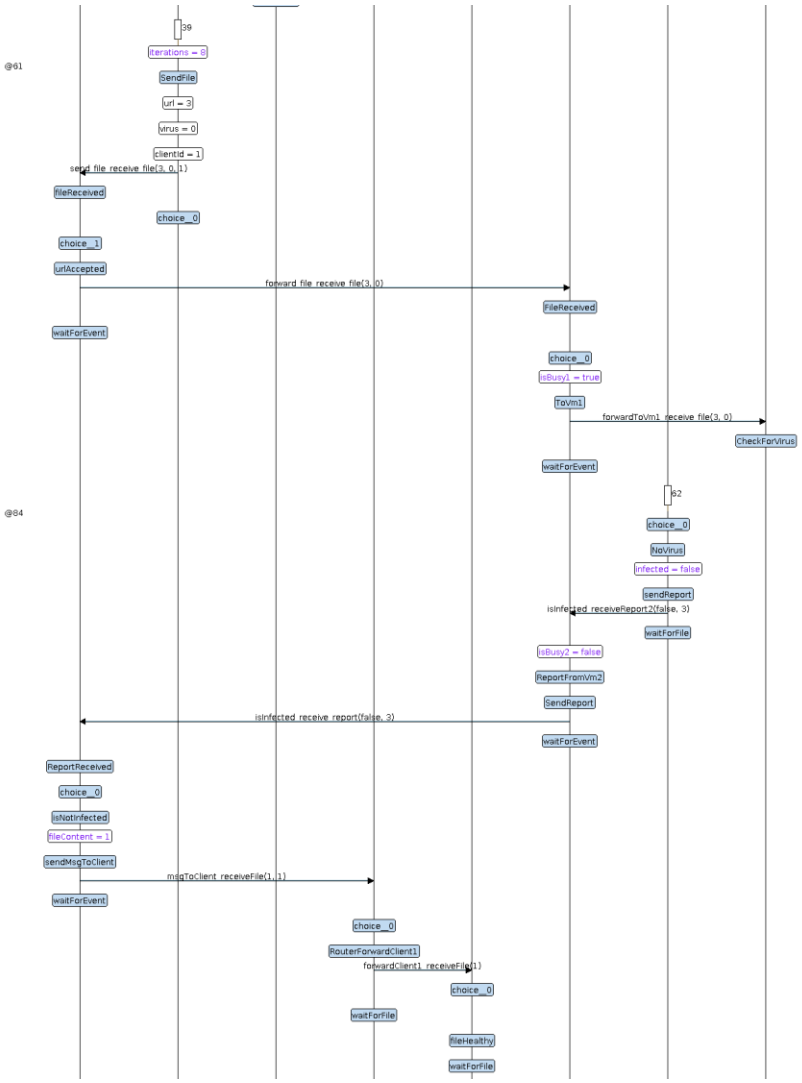

Figure 13- File sent to VM1

This trace shows a situation where VM2 is busy and VM1 is available. The received file is correctly sent to VM1 (at time 61).

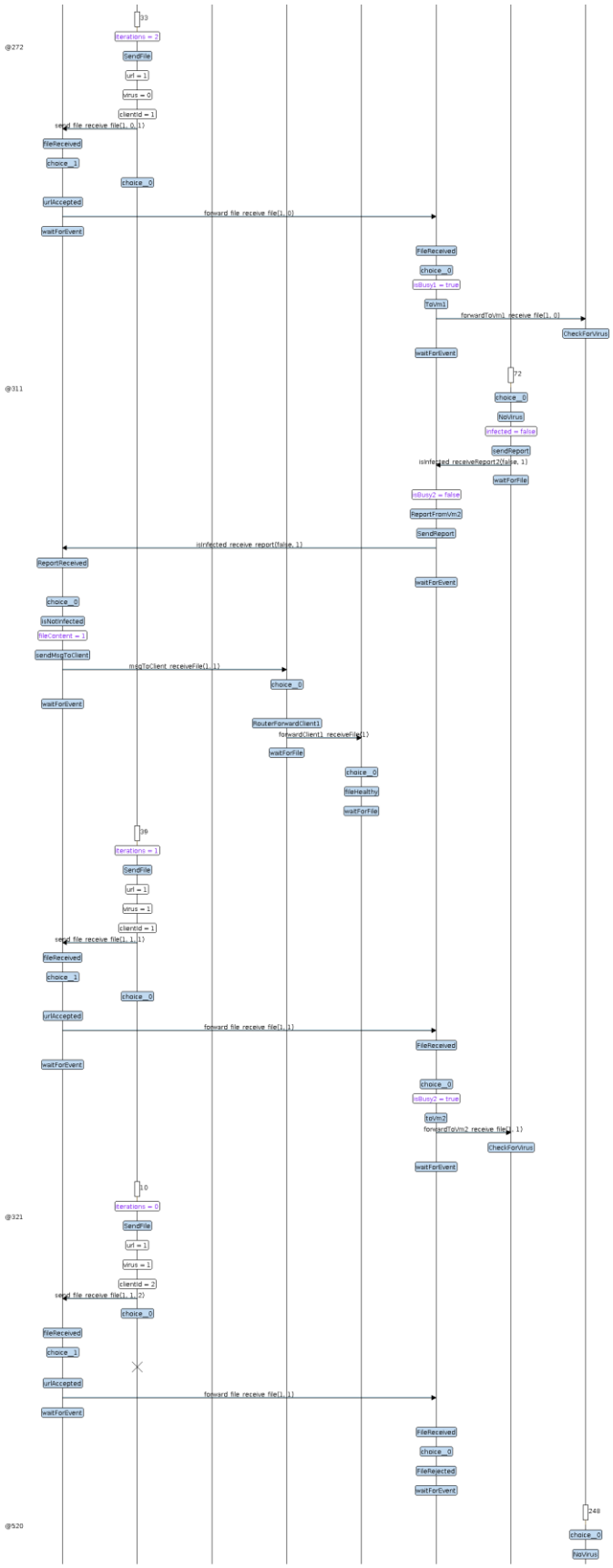

Figure 14- Both VMs are busy

This trace shows the situation where both VMs are busy. VM1 receives a file at time 272, and VM2 at time 311. When a third file is received at time 321, both VMs are busy and the file is rejected.

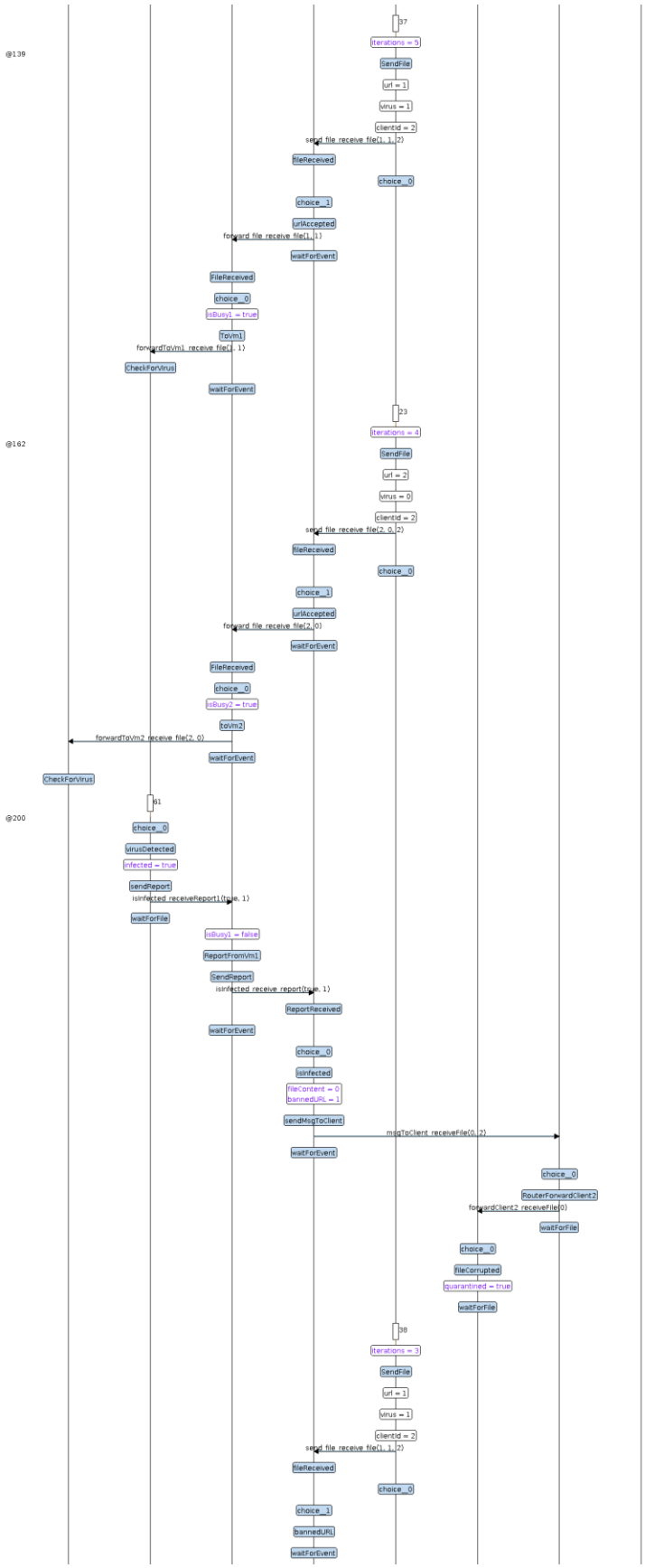

Figure 15- Trace with a banned URL

In this trace we see the process of a banned URL. A malicious file from URL 1 is received at time 139. The report alerting the maliciousness of the file is received by the FortiGate at time 200, at that point the URL is saved as the banned URL. When another file from URL1 is received at time 238, the banned URL is recognized, and the file is immediately rejected.

6 CONCLUSION

The solution presented in this report gives satisfying results. It effectively handles the three scenarios defined earlier. It also detects when a file comes from a banned URL. Some work could still be done on the **FortiController** to be able to buffer an incoming file when both VMs are busy. This would be the first feature we would want to add to improve the system.