5 Testing

This project primarily focuses on software, therefore a significant amount of our testing will revolve around this. The goal of this project is to: fill a server's cache while collecting timing data, determine the server's cache size, analyze the cache for misses, and compare this data against Wonderless dataset to determine what function was being run on the server. The code must be able to be compiled by the Firecracker environment as well as perform the vital timing and cache filling functions required for the project. Each programming language is unique and therefore each will have unique testing.

5.1 Unit Testing

Only certain programming languages are allowed to run in the Firecracker environment, therefore we had to find libraries in specific languages to fulfill certain project requirements. A critical requirement in which unit testing is a necessity is timing. The goal of the project is to detect differences between cache miss and hit times between cache layers and physical memory. The closer layer caches will have significantly shorter response times than memory further from the CPU, therefore we must test to verify that our timing libraries chosen are within an acceptable range (nanoseconds ideally). Multiple tests, with the same dataset, will occur to determine a possible standard deviation.

Understanding how much cache memory our server contains will require testing/research. This occurs by checking the values of the cache response times as the system runs our code. Once the amount of cache memory has been determined; we will need to run the code parameterized no by the amount of cache memory available. To test this we will do a similar procedure as above and make sure our fetch times are not that of the cache levels above or below our test mark.

5.2 Interface Testing

This project takes place via the cloud, therefore we will have to interface with the WLAN, before reaching our virtual environment of Firecracker. Testing this requires the team to attempt communication with the server. Communication with the server means a connection could occur, if wanted, as well as code that can be sent and executed in the Firecracker environment.

Multiple network checks, such as a ping, could occur to make sure the simulated cloud environment is working properly in regard to the ability to communicate. These network checks make sure that there are no issues on the server side as well as on the client side.

5.3 Integration Testing

Maintaining commented and legible code will reduce integration pains with this project. This project primarily focuses on the execution of code, interpreting the results, then comparing the results to a large database, but the code must also work within the given Firecracker environment. Successful integration means the attack code will run properly in the Firecracker environment. Running properly assumes that the attack code can determine cache hits and misses within a reasonable time deviation. Testing occurs in an Agile environment, meaning that testing and development occur often. Since Firecracker has a list of potentially viable languages to use, each member of the team will select one and attempt to exploit a potential side-channel vulnerability. Successful integration means our attack code can execute successfully in Firecracker and produce the timing results as well as cache hits/misses as intended.

5.4 System Testing

For our project, one of the major system level testing requirements is that the running code remains within the constraints provided by cloud providers. While we will be using our own hardware for the testing, cloud providers publish the images that they use for people to build against. Using this, we are able to emulate the cloud environment, and test against it. In these tests we can measure the time and space requirements provided to us, and verify that our code is staying within them. Running this code locally also will allow us to segment the code that we have running, and test each function individually. This allows us to individually identify what sections of the code that we need to improve, or that are preventing our code from completing in the required constraints.

5.5 Regression Testing

To ensure any new additions do not break the old functionality we will be sure to save functioning code both locally and on the server. This ensures that even if it is deleted locally or on the server we will have a backup. We are also using Gitlab, so any functioning code will be pushed to Gitlab and we can always recover it. The critical features that we need to ensure do not break are our server and to ensure our Micro VMs and code are working properly. The server is not entirely in our control but is a very important component of our design. We need to ensure that the server is working properly during all the testing of our coding languages.

Our design is driven by requirements. We are required to make sure that our code is written properly and consequently working as expected. We are also required to make sure that we are collecting accurate data and that our server is running as it should.

5.6 Acceptance Testing

We will demonstrate that our functional and non-functional requirements are being met by meeting with our client often and by collecting data that is useful and shows a difference in security based on the interpreted language being used in a serverless environment. Since our client

is also our point of contact it is important that we meet often to discuss project goals and to get help when an issue arises. Effective communication with our client/ point of contact will give us many opportunities to demonstrate that our requirements are being met. So far we have started to demonstrate how our code is functioning and collecting data. These demonstrations will continue as the school year progresses.

5.7 Security Testing (if applicable)

This doesn't apply to our current efforts to read cache information, but if we determine that implementing our research in real-world environments merits efforts to fix any exploits we use, we'll be doing security testing almost definitionally.

5.8 Results

Right now we are still working to complete the testing environment, and test cases. As we are building this, we intend to integrate with GitLab CI/CD pipelines to automate the process of testing the new code upon it being created or modified, instead of having to constantly rely on manual testing. This will be done with the GitLab runner software, to go through the process to build the development environment, and running the code, reporting the results back to GitLab automatically, and sending notifications to the team anytime there is a failure.