4 Design

4.1 Design Content

Our project is testing interpreted languages used in serverless functions for potential data leakage from other serverless functions. This will require first identifying that we are sharing a server with another function, then attempting to extract data from it.

4.2 Design Complexity

This project combines different computer engineering and computer science theories to form a unique challenge. There were several constraints put forth in this project that helped narrow down the scope, but left us with little wiggle room.

1. Memory Caching - In an effort to potentially gather information in our attack, an understanding of how memory caching works and the different algorithms a system could potentially implement. These algorithms include: least recently used, first-in first-out, etc..
2. Programming - Low level programming languages have the ability to communicate with the hardware level of a computer system, but not every system will accept their usage. We had to find suitable programming languages that could work with Firecracker, but also give us the ability to gather information about the hardware. Some examples include: Python, Rust, Go, etc...
3. Statistical Analysis - The attack we are attempting to use will require analysis of the timing it takes to gather data from each cache location. We know that each layer in the cache will take, X, amount of time, so this will tell us when we have cache misses in memory. Using this information we will create a chart that will reflect the cache misses. Time will be reflected with varying shades of color, then this visualization will be compared to a main database for any matches of known graphs.
4. Machine Learning Modeling - Once we have our data collected and labeled, we will be using it to create a machine learning model to classify future data. This will be a technical challenge for us, as we have little experience of how to do this or what kind of model would be best suited for this application

4.3 Modern Engineering Tools

Server: To emulate the cloud environment in which a serverless function would run on, we needed to have a server set up. The only consideration for this was that the server was able to support the Firecracker software.

Firecracker: This was one of the constraints to our project, so we needed to use this service. Firecracker is a Kernel-based Virtual Machine which allows for fast execution speed. The role for this technology in our project is to support Lambda, which runs serverless functions for clients.

AWS Lambda: Lambda is a Software as a Service provided by Amazon Web Services. Lambda is used to execute code quickly and efficiently via the cloud. This allows users to use the cloud's resources rather than their own.

Programming Languages: Python, Go, Ruby, and Perl will be used in many aspects of this project. We will use these languages to determine the size of the system's L1, L2, and L3 cache. The programming languages we decided to use had to be compatible with AWS Lambda and also provide the tools necessary to impact the system's hardware.

4.4 Design Context

The project will have an effect on all companies and users of serverless functions and specifically the software Firecracker. This will help the companies understand the potential issues that could come out of serverless functions, like potential vulnerabilities or data leakages.

List relevant considerations related to your project in each of the following areas:

Area	Description	Examples
Public health, safety, and welfare	This could affect the general safety of the data that users of serverless functions expect to have	If we leak information from other functions, then this could also be happening in the cloud today, and leaking private information from production code.
Global, cultural, and social	Not applicable	Not applicable
Environmental	Not applicable	Not applicable
Economic	This could have an economic impact on users of serverless functions. If there is an issue that makes them no longer feel safe enough to use these functions, then they could have added the cost of running locally instead of in the cloud.	If a vulnerability is found, and a company thinks that their copd is longer safe in the cloud, they would go and buy servers, and run it themselves out of their own data center adding cost.

4.5 Prior Work/Solutions

There has been nothing done that is exactly like this before. Some things cover partial areas of what we are doing, but did not go this far. For example this talk "Peeking Behind Serverless Functions" https://www.usenix.org/system/files/conference/atc18/atc18-wang-liang.pdf, talks about finding co-residency with your own serverless functions. Along with there being various attacks designed to leak data from processors like a cache timing attack seen here https://cr.yp.to/antiforgery/cachetiming-20050414.pdf. Our project will be some combination of both of these, with the goal requiring us to combine the ideas of finding if we are running alongside another serverless function, and then attempting to leak data from it.

4.6 Design Decisions

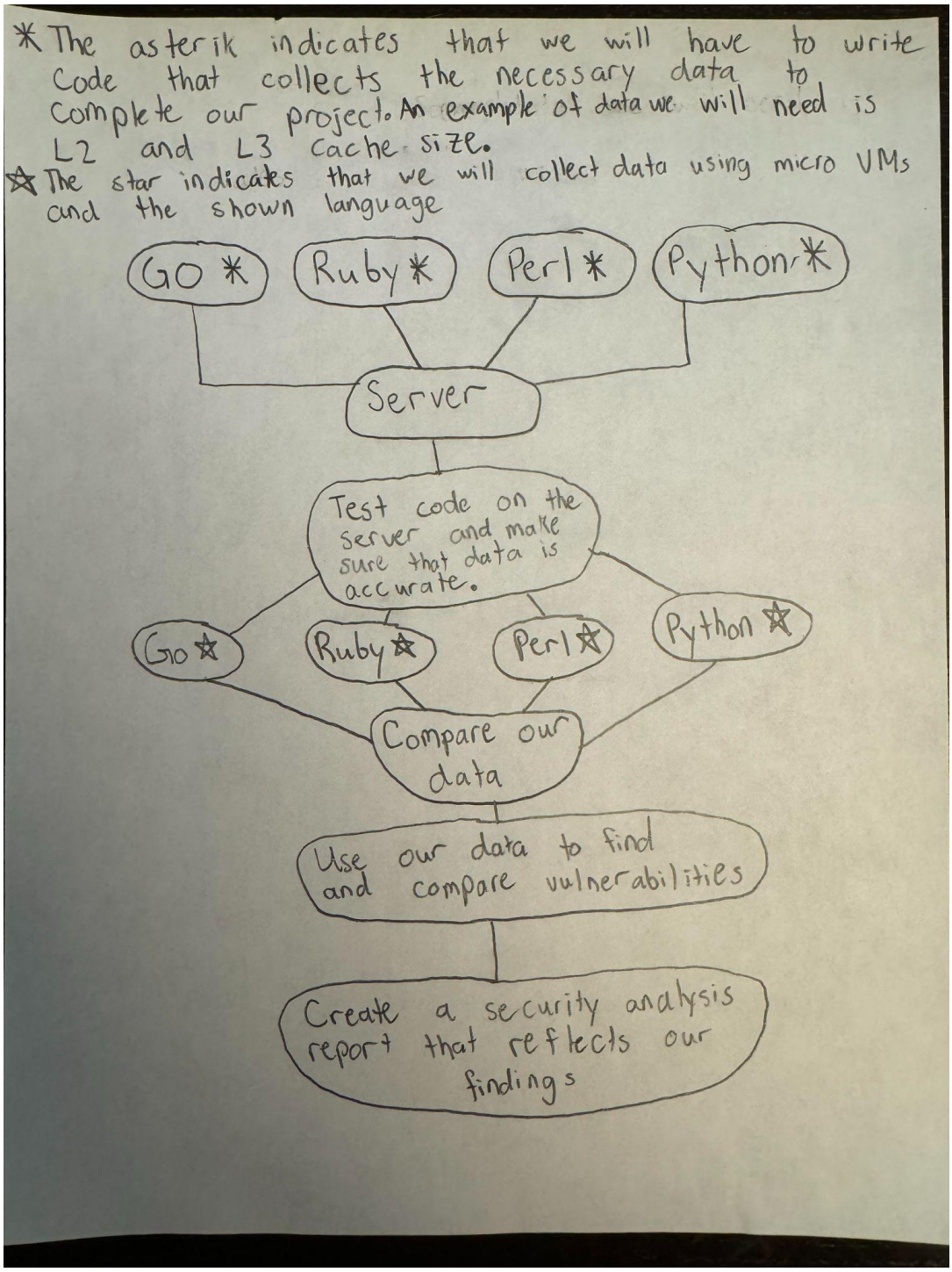
Some of the key choices that we have made so far, is that we want to have our code to run like the real world. This includes the key decision of using a local server to test the functions, instead of the cloud to prevent potentially leaking data that we do not control, along with saving time and money, as we do not

have the cloud cost. Another key decision is that we are making this server act like the cloud environment, that means running a similar setup to AWS, and running code like they would, so that if we are successful, we are more likely to need no modification when moving to the real cloud. The third constraint is just part of interacting with serverless functions. These functions have a highly limited run-time and memory constraints meaning that our attack code needs to fit within these constraints, meaning running times of less than a few minutes, and needing to use small amounts of RAM to prevent them from being removed automatically.

4.7 Proposed Design

So far we have been using multiple languages that exist in serverless functions, to test what we are able to do with each of them. This includes working to get cache timing tests, and throughput tests, that are working outside the testing environment. Moving those implementations into a firecracker VM to confirm that they are still working. After this we are going to work on scaling up the amount of data that we collect, this data that is collected will be used to train a model that identifies this data. This will allow our functions to stream data to the model, which will allow for quick classification with less data collection. This allows us to stay within the thigh timing and memory constraints.

4.7.1 Design 0 (Initial Design)



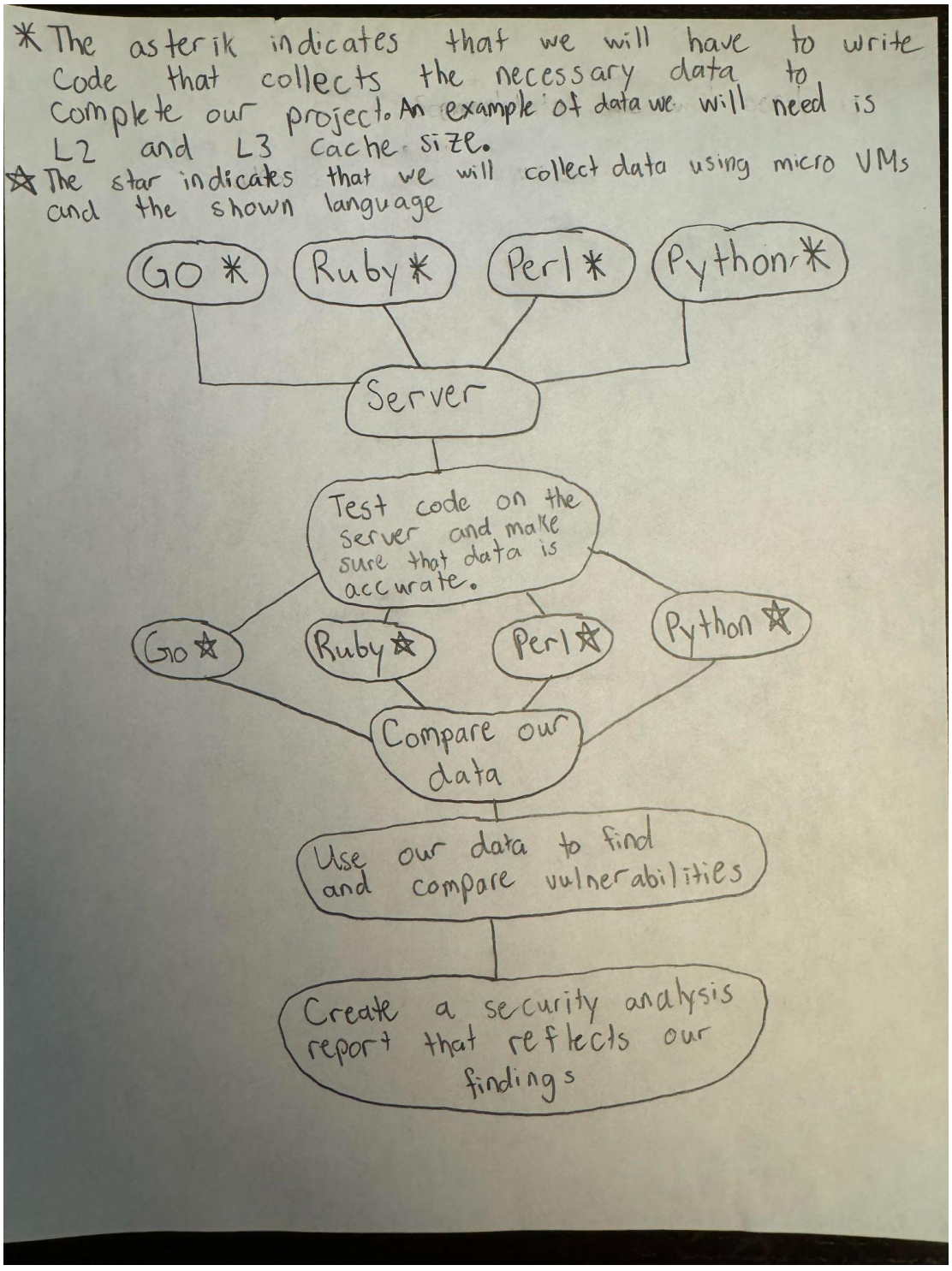
Shown above is a sketch of how our project components will all connect with each other. Since there is no physical hardware to be used in our project, this design was less traditional then some other designs that might have physical components working with each other. This design is essentially trying to show that we will collect data about a machine using the interpreted languages: Go, Perl, Python, and Ruby. Then we will test the code we have written on a server that will have a large number of micro VMs. We will compare data

and vulnerabilities that come up based on the interpreted language being used. Finally, we will create a security analysis report that will show our findings and can be understood by anyone with some sort of computer science or engineering background.

Functionality

Since our project is similar to a research project, the functionality is not entirely known. The data we collect could be inaccurate which would cause a lot of problems in creating our final report. However, ideally we will write code that works properly, and use that code to collect accurate data about different VMs. We have more non-functional requirements since our data relies on efficiency, usability, and reliability. Assuming our code, server, and VMs function as we hypothesized they will, we will be able to create a report showing the pros and cons of using interpreted languages in a serverless environment.

4.7.2 Design 1 (Design Iteration)



Design Visual and Description

Since this is the first design we have created, we have not made any changes to this design. We are certain that as the school year progresses we will make necessary changes to this design.

NOTE: The following sections will be included in your final design document but do not need to be completed for the current assignment. They are included for your reference. If you have ideas for these sections, they can also be discussed with your TA and/or faculty adviser.

4.8 Technology Considerations

Highlight the strengths, weaknesses, and trade-offs made in technology available.

Discuss possible solutions and design alternatives

4.9 Design Analysis

- Right now we are optimistic that our design will work well, and allow us to be successful.