Determinating Mandelbrot:
Adding Provider-Enforced Deterministic Execution to the Cloud

Summary

The idea for this project comes from the paper “Determinating Timing Channels in Compute Clouds” by Amittai Aviram, Sen Hu, and Bryan Ford of Yale University, and Ramakrishna Gummadi of the University of Massachusetts Amherst. There is a link to it in the Outside Resources section of this document. The paper outlines a method of protecting against timing channel attacks in the cloud by using what the authors call provider-enforced deterministic execution. It is my job to implement this defense using the research kernel Determinator, developed by DEDIS group at Yale.

The basic idea is that you may trust the cloud service provider (e.g. Amazon), but you don’t necessarily trust fellow customers. If you are running on the same hardware as an adversary, it leaves you open to timing channel attacks. For example: Alice runs a cloud compute service where you can buy space and compute power on one of her machines. Bob opens an account with Alice’s service in order to process some private financial data for his company, ACME. Eve also gets an account with Alice’s service, and happens to get placed on the same machine as Bob (If you think this is a poor assumption, read the paper “Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds,” which is also in the Outside Resources section of this document. The authors show that it is feasible to get placed on the same hardware as an adversary with fairly good probability.). Bob, who is none the wiser, starts the very computationally intensive process of crunching his company’s numbers. Eve, who wishes to discover information about ACME’s finances, monitors the cache and CPU behavior of the machine that she and Bob share, and is able to glean some meaningful information from the results.

The proposed solution to this problem is as follows: Cloud service providers such as Alice provide gateways, to which users submit job requests and all the necessary inputs for that job. The gateway sends the job off to be processed, and returns the results to the user. But the result is solely a function of the input. This means that Eve cannot possibly learn anything about Bob’s job, because she must submit explicit inputs, and her result will depend only on those inputs, and not any timing information from Bob.
For my project, I will implement this model. There will be a gateway server running a web app which packages up user input and sends it to the Determinator kernel to perform the actual computation in parallel, and then the kernel will send the output back to the gateway to be displayed to the user.

The web app will be a Mandelbrot set viewer, and the user will be able to pan left, right, up, and down, zoom in and out, change the colors of the background, and increase/decrease the resolution. These are the specific job inputs. The gateway will then send these inputs to the Determinator kernel via raw Ethernet packets (Determinator currently has no TCP/IP stack), and Determinator will calculate (in parallel) the next iteration/area to display of the Mandelbrot set, package it up as a JPEG, and send it back to the gateway to be displayed.

I have chosen a Mandelbrot set viewer to make things interesting. Some sort of computationally intensive manipulation of sensitive data probably would have made more sense given the context of timing channel attacks, but frankly the implementation would not be very exciting or interesting. Calculating the Mandelbrot set can become computationally intensive as the user zooms in closer and closer to the edge of the set, so this will actually serve as a good test in that regard. The Mandelbrot set computation can also be made extremely parallel, which makes it an excellent application to run on Determinator.

**Deliverables**

I. **Gateway**

The gateway will be a web application running on an Ubuntu box with an Apache server. The application will be a simple Mandelbrot set viewer that allows the user to pan and zoom. The goal is to have an interface reminiscent of Google maps, and each time the user zooms or pans, a request will be sent to the Determinator kernel, the next iteration/area of the Mandelbrot set will be computed, and the JPEG will sent back to the Gateway to be displayed to the user.

II. **Communication**

A method of communication between the gateway and the Determinator kernel must be implemented. Right now there is no TCP/IP stack in Determinator. Rather than add one, it seems simpler to implement communication via raw sockets. A protocol must be designed for this communication, allowing the gateway to send commands to the back end Mandelbrot implementation. During development, Determinator will be running on virtual Qemu machines, and communication will occur over the TAP interface.

III. **Back end**

The Mandelbrot set back end will be written for the Determinator kernel. Much of the C library has been ported to run on Determinator, so the back end will be written with this limited subset. The back end will receive commands from the web app (e.g. pan left, pan right, zoom in, etc.), calculate the next iteration of the Mandelbrot set in parallel, package it up into a JPEG, and send it back to the gateway to be displayed to the user. In order to create a JPEG, I will have to port libjpeg (or at least some of it) to run on the Determinator kernel.
There will be two main parts of the back end implementation:

1. **Background process**
   There will have to be a process running at all times to save the state of the current Mandelbrot view, so that it can be updated by user input. When Ethernet packets with work requests come in, the root process will spawn computation processes, wait for them to finish, package the results into a JPEG, and send it back to the Gateway.

2. **Computation processes**
   Computation processes will compute the rows of the Mandelbrot set in parallel, and return the array of pixels computed to their parent process. This is a fairly simple computation, but it gets more intensive as the user zooms in closer to the edges of the set. These computation processes will be migrated to other Determinator nodes, making use of Determinator as a distributed system.

**Outside Resources**

http://cseweb.ucsd.edu/~hovav/dist/cloudsec.pdf