CS 3410, Project 4 Design Document

Luis De Medeiros (lsd53), Ajay Gandhi (aag255)

May 6, 2015

Introduction

This document explains the thought-process and reasoning behind the design decisions taken for the project. It takes an in-depth view at implementation plans for the project and covers design choices (e.g. interrupts vs polling).

Packet Handling Approach

We decided to use a polling system to handle the packets. This approach requires extra communication between cores, which will be covered later in the document.

In addition to polling, we plan to take advantage of the synchronization primitives learned in class to create a parallelized packet handling system. The first core takes packets from the ring buffer and places them into a queue. Each core, in parallel, removes a packet from this queue when available and analyzes the packet, updating the appropriate statistics or executing the command.

Why Polling?

Polling has several advantages over simple interrupts. The most important of these advantages is the ability to handle a large burst of packets. In an interrupt-based system, a core handles packets as soon as they come; that is, as soon as a packet arrives, the core attempts to hash it, classify it, and update corresponding statistics. This entire process is time-consuming, so during a large influx, the majority of packets would be dropped.

Additionally, the polling approach makes it more straightforward to take advantage of multiple cores, since packets are not necessarily handled immediately when they are received.

Datastructures

Two main datastructures will be used to implement the honeypot.

The first datastructure is a simple hashtable, almost identical to that created in Homework 2. There will be multiple instances of the hashtable; each will be used to keep track of a its own type of packet. For instance, one hashtable will be dedicated to keeping track of evil packets and their hashes. Because of the parallelized nature of the system, the hashtable must be synchronized; that is, only one core should be able to write at a time.

The second datastructure is a synchronized queue. This datastructure is used to keep track of packets between cores. Because the cores act in parallel, only each core should be able to pop a packet off the queue; this is the only operation for which the queue should be locked. Additionally, only one core will be pushing to the queue (the rest will be popping). This datastructure is especially significant because it will allow the system to handle large bursts of network activity - it can simply write each packet to the queue, which is capable of holding a much larger number of packets than the ring buffer.