**Intro:**

In this project a simple intrusion detection system was built, with the help of functions and classes implemented in the previous two projects. The major addition that was made in this project were three classes: SignatureParser, Signature, and SignatureOptions. As can probably be predicted from the names, these classes handle the parsing of signatures, and in comparing packets to them. The class SherlockIDS was also built, and it would handle the parsing and transferring of packets to fragment handlers, if needed, or to the previously defined classes which would check for a matching signature. The class FragmentAdministrator was created in project 2, to handle the aggregation of reassembled fragment packets. In this project the added functionality of checking whether the reassembled packets matched signatures was added to the FragmentAdministrator. The below diagram shows the infrastructure that was created for project 3.



**Main:**

Main for this project served two purposes. The first was to create an instance of the SignatureParser class and pass it the name of the file that contains the rules. The other was to initiate SherlockIDS with the instance of the SignatureParser class it created.

**SignatureParser:**

This class starts of by creating a treemap object, the point of this was to have a single object that would contain all the rules in a way that could be retrieved fairly easily. The treemap object had the key of type string, which would be either ip, arp, udp, icmp, or tcp for the protocols accepted, and a value of Vector<Signature> which is the class created that held the signatures. The class then continues to read the rules from the file it was passed, and calls the Signature class to parse the rules, and stores the instance of the class in the treemap object. This class also provides a function that returns the treemap object.

**Signature:**

The starting function for this class is called parse. In this function the class receives the rule as a string, parses, and then stores the first seven parts of the rule. These are the action, protocol, ip/mask of source, port or port range of source, direction, ip/mask of target, port or port range of target. It instantiates the class SignatureOptions to handle the parsing and matching of options. The way that the rule is parsed is that the string is split using blank spaces as the delimiter. The array that is received from the splitting of the original rule string is then traversed. The first string is checked to see if it is either “alert” or “pass” and stored. If neither of these values matches, then an error is returned. The second string is checked to see if it is one of the accepted protocols and stored. If it is not, then an error is returned. The third string which is the ip address target and the subnet mask, is split into two strings this is to check that the ip address has four parts to check input. If it does not, then an error is returned. The fourth string which is the port source is split with a colon as a delimiter. If an array two strings is returned, then we know a port range is being defined. This value is checked to make sure that the first is smaller than the second, and if not an error is returned and changed to any as the port. If no range is detected then the defined port is stored as a string, this is in case that any is set.

The fifth string is compared to either the value ->, or <> and Boolean value is set that defines the rule as either bidirectional of unidirectional. If not, then an error is returned. The sixth string which is the ip address and the subnet mask of the target, is split into two strings this is to check that the ip address has four parts to check input. If it does not, then an error is returned. The seventh string which is the port target is split with a colon as a delimiter. If an array two strings is returned, then we know a port range is being defined. This value is checked to make sure that the first is smaller than the second, and if not an error is returned and changed to any as the port. If no range is detected then the defined port is stored as a string, this is in case that any is set. The class then checks if the array is longer than 7, if it is it will pass a substring of the remaining part of the string to SubstringOptions to parse for options.

The function SignatureMatching defined in this class receives an IPPacketParser object which has parsed ip packet as a class to retrieve values, also the source and destination ports of the packet, and a value of portAvailable which tells the function whether the packet has a port. This comes into use when differentiating between icmp packets, and raw ip packets. The function then checks that the ip source, and destination match the subnet mask value that the rule has. If the rule is bidirectional, then the source and target are switched and checked a second time. The function printRule is used to check what values signature has stored.