Simulation of a Distributed Denial of Service

Proof of concept of a synchronized and distributed denial of service infrastructure.
Assignment 2 - EECS 3214

Thalammeherage Raj Perera
#2111429610
cse11011@cse.yorku.ca

Overview

The solution consist of three primary components. Attackers, Coordinator and a Target. Attackers a.k.a. zombies are a silent software that is installed in multiple computers. They will collectively create a TCP session to a prespecified host at an agreed upon time. Coordinator a.k.a. command-and-control server will connect to the attackers and provide them with instructions regarding the attack such as the target's address and the time of attack. The target a.k.a victim could be any process on any computer computer listening on a TCP port. The objective is to concurrently create the most number of connections to the target thus effectively draining the resources of the target and deny it's services to legitimate connections.

How It Works

The "Attacker" processes will initially idle and listen on TCP connections from the command-and-control server. Once a connection is established, the Coordinator will negotiate with the Attacker and make sure that the Attacker's clock is in synchronization with the Coordinators clock. This is important to ensure that all Attackers initiate their attack at the same time specified by the coordinator.

Coordinator will perform periodical polling and time negotiation with the connected attackers to make sure that they are active and in sync. Once given the command to attack, the coordinator will send the message to all attackers. The attackers in turn will adjust the received time if necessary (using an offset calculated by the time sync) and launch the attack. The status of the attack will be reported back to the Coordinator and attackers will idle and wait for more commands.

Architecture

The user interfacing elements of the components share many features in common. They both utilize an interactive top level command line interface (aka REPL - read, eval, print, loop) to manage user input and output.

The logic concerning communication between components are moved into a separate class and an API (Java) is exposed for manipulation. This modular approach allows scalability and portability via ‘pluggable’ user interfaces and also encapsulates the communication protocol to a centralized location. Furthermore, the separation of concerns allow graceful error handling and developer flexibility.

Communication

All communication between the components over TCP will be facilitated by a serializable java bean which acts as the common medium. Each bean will contain an “action”, “value”, sent time, and received time. For the simplification purposes, all actions will be initiated by the coordinator and the coordinator will expect a response for each sent message. Attacker is responsible for performing the requested task by the “action” specified by the incoming message and respond with the success state of the performed action.

Coordinator

Coordinator will need to be configured with the IP and port pairs of each attacker. Upon configuring a new Attacker, the Coordinator will attempt to create a TCP connection to it and will perform a time synchronization. The connection to the attacker will not be closed and will be maintained by a dedicated thread. This ensures that the coordinator will instantly be able to communicate with an attacker and also identify if the state is changed in an attacker. Coordinator will periodically poll the attacker and perform time synchronization in a background thread to compensate for network state/quality changes. Coordinator may also broadcast the target addresses to attackers and notify the attackers of the start time of an attack.

The coordinator will maintain a local database of attackers connected to it and their status.

Attacker

Attacker consists of two components, listener and worker. Listener will listen and maintain a session with a remote coordinator, responding to commands and updating the attackers configuration from remote coordinator instructions. The worker will invoked when the Coordinator instructs the Attacker to start an attack. The worker will block the listener and wait for the attack time, then initiate a TCP connection to the target. The worker will discard all data it receives from the target but will maintain the connection till its closed by the remote host.

Time Synchronization

A time synchronization protocol is used to ensure that the Attacker and Coordinator are reading the same time. Since the application is required to perform synchronous tasks in a distributed system, it is important to account for network delays and wrong clocks. The Coordinator sends a burst of TIME_SYNC packets to the Attacker multiple times, sleeping for few seconds in between. The Attacker calculates the RTT/2 between each TIME_SYNC packet and stores the latency in an array. These latency values are then filtered such that all latencies outside one standard deviation of the median latency is dropped. This is done to remove outlier latencies occured by TCP congestion and packet queueing. The remaining latencies are then used to calculate a mean latency. The Attacker can use the latency + coordinator timestamp to calculate the time difference between its local clock and the coordinator's clock.

Server

Ther Server is a dumb application made specifically for testing the Attacker-Coordinator set up. It will listen on a TCP port and on connection, will start sending random bytes for 10 seconds and then close the connection.

Possible Improvements

Thread Pooling

Currently, there is no limit imposed on the number of threads that can be spawned by a server process. While this may allow unlimited number of connections theoretically, it is greatly dependent on the availability of resources in the computer. By utilizing thread pooling, it is possible to reuse threads and utilize the resources more effectively in a situation where the server may have to handle many connections concurrently.

Better Error Handling

There are a lot of edge cases to be handled as this is a distributed set up. Thus in certain situations such as bad/intermittent network connections and misconfigured Attackers, the system may behave erratically.

Known Issues

It is impossible for the programs to calculate network delays with accuracies < 10ms due to the random nature of the network latencies and the JVM instruction handling. These issues could be avoided by using a robust language framework/techniques optimized for real-time computation and latency detection/prediction.

Installation

1. Extract the submitted zip file to a folder, let that folder be called "appRoot".

   ```
   $> unzip assignment2.zip -d ~/appRoot
   $> cd ~/appRoot
   ```

2. Compile the sources

   ```
   $> ./compile.sh
   ```

3. To start,
   a. Attacker : **`$> ./run_attacker.sh [PortNumber]`**
   b. Coordinator: **`$> ./run_coordinator.sh`**
   c. Server: **`$> ./run_server.sh [PortNumber`**

   (Replace [PortNumber] with a port number of choice.)

Usage

Attacker and Server

Attacker/Server requires no further configuration other than the **PortNumber** which must be supplied as an argument.

Coordinator

It is assumed that one or more attackers and a server instance is running before the coordinator is configured.
Upon launch use the following command to add an attacker to the Coordinator.

```
>>> ADD(<attackerIP>:<port>)
```

After all attackers are added, use the following command to update their target.

```
>>> TARGET(<targetIP>:<port>)
```

Ensure the attackers are configured properly by issuing the following command.

```
>>> STATUS
```

If any attackers are shown to be in **`ERROR`** state, then use the following command to remove them.

```
>>> DEL(<attackerIP>:<port>)
```

Use the following command to start the attack. "TimeToStartIn" denotes the number of seconds to wait before starting the attack. It is optional and if not supplied, a default value of 5 (seconds) will be used.

```
>> START([TimeToStartIn])
```