James Howard (jhoward6)
James Zhen (jzhen4)
Dhyaanesh Mullagur (mullagu2)

P2P Botnet Progress Report

What we've done:

We have compiled a list of functions that we want each client to perform. The operations we have in mind are the following:

Upload - Uploads a local file to the target client
Download - Downloads a file from the network or from a server
Execute - Execute target file at a specified time
Drop - Remove node from network and remove local files

We have also finished one of the two sample scripts (the keylogger) that can be uploaded to the client and executed. Our other demonstration script will be a DDOS script. Our current keylogger implementation, when executed, operate in the backgrounds and logs the user's keystrokes to a local file. This file can then be either uploaded on a set interval, or through the Upload operation to a specified location.

In addition to client features, we have designed our network architecture and how we want to implement it. We have started on the implementation but it is still a work in progress.

What we have left:

We still need to implement the client operations (see above). Some of these functions are dependent on the network architecture and will be implemented as the network is implemented. In addition to the client functions we also need to write our second sample script, the DDoS script. We will most likely use a flood approach where we have each client participating in the DDoS send multiple GET requests to the target server in order to overwhelm the server's resources.

The biggest part that we are currently implementing is our network architecture. Our architecture is described in detail below.

Network Architecture:

For our architecture we're using Chord P2P. When a new bot joins the network, it will try and connect to a list of known nodes. When it manages to connect, the nodes will update their finger tables with the new node. Each node will have a unique ID assigned to it based off a hash of the node's IP and port. The hash will be truncated to some standard number of bits depending on the scale of the network we want to use. Once a node has connected, it updates its local node list with the currently connected nodes.

Each node will have a hard coded copy of a public encryption key. The commander sends a code to any node in the network, with the code encrypted using the attacker's private key. This works as a digital signature to verify that only the commander controls the botnet. Depending on the type of command, the contacted node can do a few things. In the case of a mass command, such as DDOS, the node will flood its peers with the command to distribute it through the network. This can be done more selectively by forwarding to its neighbors, along with the nodes in its finger table. If the command is an individual command such as telling one node to start a keylogger, the contacted node will use the DHT and finger tables to forward the command to the correct client.

To make sure a node isn't inserted into a network with a modified encryption key, nodes will use consensus voting to make sure that the input command is actually valid before it's executed. This prevents researchers from inserting a modified key, sending commands to that node, and then having that command propagated throughout the network.