**Ways to reduce the risks of Crowds and further study of web anonymity**

By: Manasi Pradhan

**Introduction:**

In the paper “Crowds: Anonymity for web transactions”, authors Rubin and Reiter describe a particular way of maintaining the anonymity while interacting with a web server.

Now that we know briefly how crowds work, we can see what the points that can make it vulnerable are. The following section will describe the way we try to change our approach towards the traditional crowds to patch these vulnerabilities.

**Vulnerabilities and drawbacks of Crowds:**

As we can see, in the traditional crowds the Blender is the most important entity that makes the entire working f Crowds possible. But we can see that the Blender, if is the single most important entity, then is also a single point of failure. The paper describes that the Blender cannot be a single point of failure as the Blender just gives a start to the whole process. Even if the Blender fails, the working of the present Crowds will continue. The problem will arise when there is an imminent Join Commit, which cannot take place without the Blender. Even though all this is true, we still cannot say that Blender is not the single point of failure. If the Blender is under attack, the attacker can send malicious information and the communication between Blender and all other nodes can be tampered with. The Blender is in charge of giving away the keys of all the members in the crowd to the new node and vice versa. So if this is corrupted then all the communications can go haywire. The Blender can give the new node the list of only the malicious nodes in the crowd. So that the new node will have no option but connect to only the malicious nodes that are collaborating with the Blender, unknowingly. Another limitation we can discuss here is the scalability factor. If the jondos in the crowd go on increasing exponentially, we can see that the Blender can become a bottleneck. During the Join Commit, sending the keys of all the new nodes to the entire network can result in a bottleneck of resources at the Blender which can result in huge delays to complete the Join Commit.

There are three types of attackers discussed in this paper namely local eavesdropper, collaborating jondos and the end server. The sender is exposed to the local eavesdropper. But the collaborating jondos are likely to cause more damage. In the traditional crowds as we have seen, the destination address as well as the message is open to all the jondos. So if the local eavesdropper is collaborating with any jondo, the sender and receiver both are exposed.

**Proposed solution:**

The first limitation discussed is the single Blender in the traditional crowds. So this proposal will deal with the use of multiple Blenders as an effort to decentralize the system. These Blenders will each control a particular group of users connected to them. The number of these Blenders will depend on the number of users currently in the system.

Let us describe the working of this system:

Initiation process- In this process, we first need to register the users with a particular Blender so that they can be added to the system. So when the user wants to be added, it will contact any Blender out of these multiple Blenders which in turn will return the list of Blenders to the user. The user will then generate a random hash function and hash its own IP address with it.

Joining- The solution of this hash will be compared to the list of IDs of the Blenders. The closest ID to the result hash will be contacted by the user. This Blender verifies the user and accepts it into the system. It then waits for the Join Commit to take place.

Join Commit- First, the members in the crowds will send their new public keys to the Blenders. These will be used for that particular session only. So every new session will need the users to create their public keys.

Second, the Blenders will send the list of their members to each other. This list will contain all the jondos and their public keys for that session. These keys are used for Diffie Hellman key exchange that will take place when the session starts. This key exchange is necessary so that the nodes will get their respective link encryption keys that will be needed to encrypt the link during the sending of message from one jondo to another.

Next, the Blenders will send these public keys of the entire network to all the existing and new jondos.

Once, all the Blenders have the list of the entire system and all the jondos have the public keys of all the nodes, this crowd will work like a traditional crowd.

Formation of New Blender: We need to discuss when the system needs a new Blender and when will it be created. As and when the system increases there will arise the need for a new Blender.

During the Join Commit, the Blenders will exchange lists when either of the two scenarios take place:

They will wait for a particular time period and accept all the requests from the new users. After timeout, they will exchange the lists.

When they reach a particular threshold of users they can be responsible for.

When all the Blenders get all the lists they can check whether any one of them have reached the threshold. If only some Blenders have reached the threshold and they still get the requests for join, they will just refuse the entry. The user will then re hash and request another Blender.

If on checking, the Blenders realize that all the Blenders have reached the threshold and they are still getting the requests, then they willl select one random jondo among the oldest jondos and appoint it as the new Blender.

The next drawback which we discussed was the intermediate jondos being able to monitor all the traffic, the destination address as well as the message. Hence we propose a new concept of the End Jondos. The End Jondos are a layer before End Servers. The user will have public keys of all the End Jondos just like it will have of all the other jondos. The destination address of web server is encrypted with this key.

Hence, the destination address visible to intermediate Jondos will be the address of an End Jondo instead that of the End Server. This End Jondo on receiving this message will check if this message is encrypted by its own key. If the End Jondo cannot decrypt it, it will know that it is not the one which will pass it on to the End Server. So next, it will pass it to another End Jondo and another till it reaches the one who it is meant for. Once a particular End Jondo is able to decrypt this message then it will forward it to the End Server.

To see, this is a little complicated concept but it is necessary to see why it is being implemented the way it is. In this system, even if local eavesdropper sees the address, not sure if it is the End Jondo that sends to End Server as they both are different most of the times. If the intermediate Jondo is collaborating with the End Jondo, then it is likely that it will forward the message to its own favored End Jondo. Instead now, the End Jondo is the one user trusts or user wants.

**Limitation of the current proposed system:**

There are still many issues that are not answered at all or are not being addressed efficiently in this new system as well. The main issues are what if the blenders are corrupted? They can easily send the wrong public keys. This can be solved in some way but it not likely to be very efficient. One solution can be that we can make all Blenders keep track of all the public keys and broadcast them. Another one can be that, the Blenders can use a function to send the list to only particular users of the Crowd to avoid too much traffic. Thus this can allow the users to compare and consider the majority value. But again, if the majority jondos are corrupted then the majority value will be corrupted as well which will lead to inefficient behavior of the system.

Another question is whether the active attacks possible. The message in encrypted and the address is encrypted with unknown End Jondos public key. Hence, when a message is received by a Jondo it cannot see the destination address, the message or the sender address. But in this too much importance given to the End Jondo which again makes a lot of information to be vulnerable. So, active attacks can still be launched by the End Jondos who can decrypt the messages.

Next we can see that the attacker has to control more than half Blenders in order to pass the false information to users. For the users to be able to cross check the information with the other Blenders maximum Blenders need to stay uncorrupted. Also the question arises that how many Blenders maybe enough to verify from because verifying from allt the Blenders all the time will seriously hamper the scalability and the traffic of the system for which the multiple Blenders were created in the first place.

**Inferences:**

We can see that a lot of work is still left to do on this system. There are some inferences that can be drawn from the work above some of which need to be worked on hopefully in the future:

We did improve in some cases. We decentralized the system to avoid single Blender to be corrupted by the attacker. We have multiple Blenders instead. But now too much additional traffic is being injected in the system- exactly how much, is it a considerable tradeoff? Also, nothing added to stop the timing attacks. We can add extra jondos in the system through which the traffic maybe routed dynamically- like free routes.