**Mix Network Intersection Attack Simulation and Analysis**

# Abstract

This paper describes a mix simulator and its use in examining the factors which determine how many messages a sender remains anonymous for during an intersection attack. The simulator and resulting analysis assume a global passive attacker, and all honest mixes. By developing the simulator and using it to run experiments, insights may be gained into the GPA’s per round progress. These insights will be used to formulate a function which describes the attacked sender’s anonymity set size after M messages.

#

# Problem Defined

Given input, can an Anonymity Set of S size be guaranteed for N messages? What input is necessary? Can it reasonably be obtained?

To address the above questions, the following steps were planned:

1. Develop software to simulate an intersection attack give the input parameters. The simulation returns the attacker’s view of the sender’s anonymity set after the number of messages has been simulated.
2. Analyze results from Part 1 results to formulate a function which returns the number of mixes needed given the input parameters to ensure a sufficient anonymity set.

#

#

# Simulator Design Specifications

1. Allow configuration of mix network structure, including degree of “cascadeness”.
2. Allow a minimum acceptable anonymity set size to be specified.
3. Show the number of messages the sender sends before the GPA reduces the sender’s AS size to ASmin.
4. Depict the mix network structure in an accessible manner.
5. Show how messages flow through the network.
6. Generate ambient message traffic.
7. Show the intersection attack in progress.
8. Clock the simulation, such that it can be run one tick at a time.
9. Run multiple simulations using the same program instance; remember the last parameters used.

#

#

# Simulator Implementation

MixSimulator was developed as a windows 10 universal app in C#/Xaml. It utilizes standard MVVM architecture and only standard libraries. In its final release it was distributed as a Visual Studio 2015 project.

**UI**



##

##

**Model**

Internally, the mix network was represented by a graph of nodes managed by an instance of a Network object.



Each Node contained a Network object and references to the nodes it was linked to. Mixes and Users were subclassed from NetworkObject, and overrode what was done upon sending or receiving a message. Messages were simply modeled as containing a sender, receiver, and content. Whenever the mix network was displayed, link objects were constructed from it to represent the lines which would visually connect the NetworkObjects.

 Several key algorithms require further elaboration:

1. Mix configuration: Each rank is assigned 1 of the first n mixes. An error is thrown if there aren’t enough mixes to fill all ranks. Afterwards, remaining mixes are randomly distributed across ranks.
2. Targeted sender/receiver: Before a simulation is run, a targeted receiver must be designated by the user. Behind the scenes, a sender is chosen at random to have interesting messages, and be subjected to the intersection attack.
3. Node connection generation: Each sender randomly connects to 1 rank0 mix. Each mix randomly connects to at least 1 lower rank mix (or sender) and at least 1 higher rank mix (or receiver). Each receiver connects to at least 1 max rank mix.
4. Message flow: Each tick, the targeted sender sends a message. Each other sender has a 50% chance to also send a message. Upon receiving a message, a network object takes appropriate action (ex. store/flush) and forwards the message to the next node, until the message reaches its receiver.
	1. Next node to receiver: At any point in a message’s path, the message’s current node’s FromLinks are evaluated sequentially on whether or not they can ultimately be followed to the receiver. The first link that can is taken.
5. Intersection attack: The links actually used in a round as flagged as used. All others are discarded. The inverse from 4.a occurs, as the GPA determines if each receiver ultimately links to each not yet eliminated sender.

##

## **View**

 

Due to the lack of an appealing (and affordable) diagramming library, the network display portion of the app’s UI was constructed using existing components and custom helper classes. Shortly after the search for a diagramming library failed, the design decision was made to build the display primarily from ListViews. Necessary simplifying design decisions followed.

The mix network would not be displayed as a free flowing graph with draggable, editable, and addable/removable components. Instead, it would be build around columns, with the mixes organized by ranks.

Linking components visually proved extremely difficult, but the challenge was gradually overcome as helper classes were used to extend existing functionality. A LineHelper class was created to abstract away the process of connecting two FrameworkElements with a line. Unfortunately, the ListView API made it prohibitively difficult to access the FrameworkElement which represented a given Mix or User. Thus, a VisualTreeHelper class: GetFrameworkElementFor(ListView lv, Item item). Race conditions and other strange timing bugs interfered, but were resolved by applying the awaitable pattern and popups between tasks such as displaying links and clearing them. Very late in development, a binding hack was implemented to force senders to turn red when eliminated.

**ViewModel**

The ViewModel simply contained the driving logic of the simulator.



1. InitNetwork(): Generates a network from the input parameters. Effectively resets the application’s state with the current input.
2. SimulateTick(): Simulates a round in which the sender of interest sends a message to the targeted receiver, and all other senders have a chance to send a message to a random receiver.
3. RunSimulation: Effectively, runs ticks (simulates rounds) until the intersection attack succeeds.

#

#

# Experiments

 Experiments conducted using the MixSimulator were focused on exploring the impact of the number of senders, mix/cascade configuration, and threshold.

1. Senders

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| #Senders | #Mixes | MaxRank | MixType | Threshold | #Receivers | ASmin | Message |
| 4 | 4 | 3 | Threshold | 1 | 4 | 0 | 4 |
| 4 | 4 | 3 | Threshold | 1 | 4 | 0 | 2 |
| 4 | 4 | 3 | Threshold | 1 | 4 | 0 | 3 |
| 4 | 4 | 3 | Threshold | 1 | 4 | 0 | 5 |
| 4 | 4 | 3 | Threshold | 1 | 4 | 0 | 2 |
|  |  |  |  |  |  |  |  |
| 8 | 4 | 3 | Threshold | 1 | 8 | 0 | 4 |
| 8 | 4 | 3 | Threshold | 1 | 8 | 0 | 3 |
| 8 | 4 | 3 | Threshold | 1 | 8 | 0 | 3 |
| 8 | 4 | 3 | Threshold | 1 | 8 | 0 | 6 |
| 8 | 4 | 3 | Threshold | 1 | 8 | 0 | 3 |
|  |  |  |  |  |  |  |  |
| 16 | 4 | 3 | Threshold | 1 | 16 | 0 | 3 |
| 16 | 4 | 3 | Threshold | 1 | 16 | 0 | 5 |
| 16 | 4 | 3 | Threshold | 1 | 16 | 0 | 7 |
| 16 | 4 | 3 | Threshold | 1 | 16 | 0 | 4 |
| 16 | 4 | 3 | Threshold | 1 | 16 | 0 | 6 |

1. Network Configuration (“Cascadeness”)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| #Senders | #Mixes | MaxRank | MixType | Threshold | #Receivers | ASmin | Message |
| 4 | 16 | 1 | Threshold | 1 | 4 | 0 | 1 |
| 4 | 16 | 1 | Threshold | 1 | 4 | 0 | 2 |
| 4 | 16 | 1 | Threshold | 1 | 4 | 0 | 1 |
| 4 | 16 | 1 | Threshold | 1 | 4 | 0 | 1 |
| 4 | 16 | 1 | Threshold | 1 | 4 | 0 | 2 |
|  |  |  |  |  |  |  |  |
| 4 | 16 | 3 | Threshold | 1 | 8 | 0 | 3 |
| 4 | 16 | 3 | Threshold | 1 | 8 | 0 | 2 |
| 4 | 16 | 3 | Threshold | 1 | 8 | 0 | 4 |
| 4 | 16 | 3 | Threshold | 1 | 8 | 0 | 2 |
| 4 | 16 | 3 | Threshold | 1 | 8 | 0 | 3 |
|  |  |  |  |  |  |  |  |
| 4 | 16 | 7 | Threshold | 1 | 16 | 0 | 3 |
| 4 | 16 | 7 | Threshold | 1 | 16 | 0 | 5 |
| 4 | 16 | 7 | Threshold | 1 | 16 | 0 | 3 |
| 4 | 16 | 7 | Threshold | 1 | 16 | 0 | 3 |
| 4 | 16 | 7 | Threshold | 1 | 16 | 0 | 6 |

1. Thresholds

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| #Senders | #Mixes | MaxRank | MixType | Threshold | #Receivers | ASmin | Message |
| 4 | 16 | 7 | Threshold | 2 | 4 | 0 | 4 |
| 4 | 16 | 7 | Threshold | 2 | 4 | 0 | 3 |
| 4 | 16 | 7 | Threshold | 2 | 4 | 0 | 3 |
| 4 | 16 | 7 | Threshold | 2 | 4 | 0 | 3 |
| 4 | 16 | 7 | Threshold | 2 | 4 | 0 | 4 |
|  |  |  |  |  |  |  |  |
| 4 | 16 | 7 | Threshold | 4 | 4 | 0 | 4 |
| 4 | 16 | 7 | Threshold | 4 | 4 | 0 | 6 |
| 4 | 16 | 7 | Threshold | 4 | 4 | 0 | 6 |
| 4 | 16 | 7 | Threshold | 4 | 4 | 0 | 5 |
| 4 | 16 | 7 | Threshold | 4 | 4 | 0 | 8 |
|  |  |  |  |  |  |  |  |
| 4 | 16 | 7 | Threshold | 8 | 4 | 0 | 8 |
| 4 | 16 | 7 | Threshold | 8 | 4 | 0 | 8 |
| 4 | 16 | 7 | Threshold | 8 | 4 | 0 | 9 |
| 4 | 16 | 7 | Threshold | 8 | 4 | 0 | 5 |
| 4 | 16 | 7 | Threshold | 8 | 4 | 0 | 8 |

#

# Analysis

**Experiments**

The number of senders increasing resulted in more messages being anonymously sent. Predictably, with all else held constant and a Ps of 50%, the number of anonymously sent messages increased by 1 with each doubling of the number of senders.

 Higher resemblance towards a cascade configuration resulted in more messages being safely sent.

 Higher threshold resulted in more messages being safely sent (however, it was observed that sender’s anonymity set decreased drastically when key mixes flushed, rather than gradually).

**General**

 While constructing the simulator, several key observations arose. Traffic by other senders is unpredictable. Therefore, a guarantee is impossible without active measures (dummy messages). However, if average traffic is known with respect to targeted sender (as in the simulation), approximations can be made.

Let Ps be the probability of sender S in Stargeted’s AS sending a message in the same round Stargeted sends one. Then:

Entropy = - Sum Ps log2(Ps)

2^Entropy = AS for next round.

AS for round R = 2^Entropy where Ps,r = Ps^r

f = NumSafeMessages(|AS|min, AS, Ps) = ?

g = |AS|min(R) = 2^[- Sum (Psn^R) log2(Psn^R)]

f = g-1, given EAS = AS

Knowing enough Ps’s and enough about network configuration (relation of number of messages to rounds, that parallel paths are impossible) can provide a conservative estimate of AS for a message… but that’s not practical! The only party that really knows a sender’s AS on a mix network is a GPA.

# Conclusion

Given input, an anonymity set of size S cannot be guaranteed for N messages. Dummy messages must be used so that sender’s cannot be eliminated during an intersection attack. However, knowing the probability of other senders sending a message in a round can provide an estimation of the GPA’s view of the anonymity set.

Unfortunately, that information is not practically obtainable.

 It was observed that the most useful links for an intersection attack during simulation (particularly with low thresholds) werethe links between senders and rank0 mixes. This is because most senders were eliminated due to lack of sending a message during a round, rather than the route taken to receiver.

# Concluding Remarks

 The project has numerous possibilities for extension. Many parameters such as untargeted senders’ chance to send on a tick, the number of possible connection for nodes to make, etc. could easily be exposed and experimented with.

Other new features could be added:

1. Maximum path length
2. Effective anonymity set size calculations.
3. Ability to designate mixe(s) as dishonest
4. Introduction of timing, hybrid, and pool mixes

 Finally, the project raised several possible questions for future study:

1. Is it viable to implement a friendly, GPA such that it can warn users when their degree of anonymity becomes below their desired level?
2. Can the risk of this GPA providing a single point of failure be mitigated?
3. Since most senders are eliminated from observing they sent NO message during a given round… would it be possible to group senders based on recent activity with regards to rank0 mixes?

#

#

# Resources

1. [Generalising Mixes](http://www.cise.ufl.edu/~nemo/anonymity/papers/diaz%3Apet2003.pdf) by Diaz and Serjantov (Mix Types)
2. [Crowds: Anonymity for Web Transactions](http://www.cise.ufl.edu/~nemo/anonymity/papers/crowds%3Atissec.pdf) by Michael Reiter and Aviel Rubin. (Degree of Anonymity)
3. [The Disadvantages of Free MIX Routes and How to Overcome Them](http://www.cise.ufl.edu/~nemo/anonymity/papers/disad-free-routes.pdf) by Berthold, Pfitzmann, and Standtke (Mix Network)