## Exercise Set 5

helge206 Andrew Helgeson
wayt0012 Ethan Waytas
maure113 Brian Maurer

1. ***For your write-up explain briefly which "target" users you chose, whom you identified as their friends, and what procedures you took to identify those friends.***

For this question we implemented the version of the statistical disclosure attack outlined in section 3.1 of the linked paper titled *"Complex senders, unknown background traffic"*. Our implementation was scripted in Python.

Given the names of our group members we targeted the users a0, b0, and e0. Below are the top two (by probability) friends we found for each of these users: (target -> friends)

a0 -> ['x3', 'i8'] with probabilities [0.9090, 0.4545]
b0 -> ['t7', 'i8'] with probabilities [0.8809, 0.4761]
e0 -> ['r4', 'i8'] with probabilities [0.9333 0.5333]

Procedure:
We followed the statistical attack outlined in the paper.

Number of rounds: 256
Number of users: 260
Number of users per round: 32

1. We first initialize background traffic and observation vectors with lengths equal to the number of users to be all 0.
2. Iterate through all send rounds.
    a. If the target user sent in a round then we loop through the users in the corresponding receiving round and set each user's entry in a temporary vector to be 1/32 if they received something in that round. After checking all receiving users we add this into the main observation vector. We also add how many messages the target user sent to a total count.
    b. If the target user didn't send in a round we go through all users in the receiving round and do the same thing as in 2.a but add the result into the background traffic vector. We also increment a counter for the number of rounds the user didn't send in (t' in the paper).
3. After analyzing all rounds there is a little bit of processing to do for these vectors. First we

divide the background traffic vector by the number of rounds the user did not send in. Then we multiply the vector by the difference between the number of users who send in a round and the average number of messages sent by a user in one round.

4. Multiply the observation vector by the number of users in each round. Next we divide this vector by the number of sending rounds observed (256).
5. Then to create the final probability for each respective value in the observation and traffic vectors we subtract the traffic value from the observation value and divide by the average number of messages sent by the target user in a round.
6. We then selected the two friends show above by taking the two highest probabilities.

2.

a) ***One possible defense against this is for the mix to wait to mix a batch until it has seen messages from K different senders. Why doesn't this work?***

If the attacker has access to K-1 machines or just K-1 different identities (depending on how the remailer defines a unique sender) the attack would still work. If the attacker had a botnet they could spam the remailer with a message from K-1 different machines with a known destination and the attack would proceed identically by finding the $K^{th}$ message's destination, which is the one not sent by the attacker.

b) **The reason that Tor has this goal (rather than a stronger one, for example f3 rather than f2) is that an adversary who controls the first and last nodes of a Tor connection can trace it, even though the packets leaving the first node are decrypted before they go to the last node. Explain how this is possible.**

The challenge to overcome in tracing a connection even when there is control of both the entry and exit nodes is to bridge the communication gap between the two when the relay node is not compromised. Because of the layered encryption of Tor, it is not possible to simply bypass the relay node or the data would be unreadable. We have found and thought of two ways to get around this problem.

One possible option to link a sender with a receiver would be to tag the data as it enters the entry router and compare the data as it leaves the exit router. This type of method would be some type of crypto-tagging, as the onion being sent would be slightly modified in some way such that the routing policy is not broken, but a distinct characteristic is now associated with it. The idea for this type of attack came from the TOR Project blog. The specific blog entry is [https://blog.torproject.org/blog/one-cell-enough].

A crypto-tagging attack would happen as follows. A compromised entry router would receive an onion and tag it. The entry node would then send out the tag information to all the compromised routers. When the onion finally made it to the final node, it would be checked to see if it contained

the tag. If it does, the sender and receiver can be linked, breaking anonymity. This solution isn't foolproof as the crypto-tagging approach can break the connection and expose the compromised routers.

When the crypto-tagging approach cannot be used to bridge the gap between the entry and exit node, there are still other probabilistic approaches to determining communication partners. By monitoring the timing and volume (both in packet size and raw quantity over time), the entry and exit nodes can put together a correlation of senders to receivers. If there is a lot of traffic noise which isn't flowing through both the entry and exit nodes it can disrupt the calculation of this correlation, but given enough time a strong statistical conclusion can be made about communication partners. Since communication responses will travel along the same path in reverse back to the sender, this provides another point of correlation. This way traffic that didn't originally enter through the compromised entry node but did go through the compromised exit node could be filtered.

3. ***One commonly proposed countermeasure to this attack is to allow each voter to cast multiple ballots, with only the most recently submitted ballot being counted. Discuss any potential attacks that could arise as a result of this procedure.***

Potential attacks:

- They could hold you in custody until the vote is over so you can't send another ballot in. If these are some really bad people they could just kill you.
- The attacker could hold your initial ballot (the one they *bought* from you) and wait to send it in until the latest possible time.
- The attacker could spam a bunch of ballots for a large number of people. The attacker would likely send out the ballots at the latest possible time; we assume the ballot is postmarked and this is used to determine the most recent submission of a ballot. In this situation, the attacker might have a list of absentee voters, or, the attacker is hoping that some of the ballots sent in will be counted (absentee ballots for non-absentee voters would be dropped).
- As above, if we are assuming that the most recent ballot submission is determined by postmark date then the attacker could fake the postmark of all fake *attack* ballots to be the latest possible time for submission. This would almost guarantee that the legitimate submitter would not have their ballot counted since they wouldn't wait that late nor fake their postmark date.

4. ***Describe a cut and choose protocol to prevent the state-level authority from "pre-stuffing" the ballot boxes. Specifically, if there are k precincts, your protocol should allow the state authority to cheat with probability at most $2^{-k}$ .***

To create a zero knowledge proof scheme the state must send two ballot boxes to each

precinct. On receiving the ballot boxes, each precinct would flip a coin and proceed to destroy a corresponding ballot box (without destroying any potential contents) . The precinct uses the box that was not destroyed to collect votes. Of course, the consequences of the state stuffing ballot boxes is assumed to be negative, such that the state would not want to be found to be stuffing boxes (maybe the UN would come in and write a strongly worded letter of condemnation). The precinct will be able to see if the destroyed box was stuffed, but they will not know if the intact box that they end up using to vote with is stuffed. With no way for the state to know which box will be destroyed and therefore revealed, the state has only ½ chance to guess the correct box to stuff.

If each of the *k* precincts carries out this method, there are *k* instances where the state would need to correctly guess which box will be destroyed. This results in only a 2^-*k* chance that the state would be able fool all the precincts with a ballot box stuffing scheme.