Interim Report

16 Onions

Josef Stark Charlie Groh

June 15, 2017

1 General

We are the team “16 Onions” consisting of Josef Stark and Charlie Groh, and our goal is to develop a prototype implementation of the Onion module.

2 Process Architecture

TODO Charlie: haben uns fr EventLoops entschieden; falls sich whrend der Implementierung Threads als gnstiger erweisen (weniger Aufwand/einfachere Modulstruktur/bessere Performance/sonstwas) schwenken wir evtl doch darauf um. Multiprocess wird aber ausgeschlossen, da es in Java unblich und schwierig zu realisieren ist, mehr Resourcen bentigt und die Isolierung bei der geringen Modulkomplexitt noch wenig Sinn macht. ausserdem noch einleitung und schluss wenn du meinst, und ich habe dir weiter unten noch ein TODO wg. bytefield markiert, das ich nicht loesen konnte.

3 Inter-Module Protocol

For the communication between distinct onion instances we decided to use both TCP and UDP as underlying protocols in order to avoid reinventing the wheel, since they both fulfill the respective requirements perfectly.

Control messages, i.e. messages for tunnel construction and tunnel destruction are transferred and forwarded over TCP, because for those messages it is very important that they actually arrive and that we get feedback if one of those messages could not be delivered to the target, so we can react in an appropriate manner, e.g. assume that the corresponding node went down and construct

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

an alternative route. TCP satisfies these requirements as it acknowledges the reception of messages, resends messages if necessary and reports a failure if a message still didn't provoke an acknowledgement after a few retries. For this it sacrifices some bandwidth and latency, but those two factors aren't of uttermost importance to control messages anyway.

User data messages, i.e. messages containing VoIP data, are transferred and forwarded over UDP, since for those a short delay is a requirement which UDP can satisfy. UDP is packet based and does not check the arrival of packets at all, so the lower delay that this causes comes at the price of possibly losing some packets which are not resent and therefore never reach their target, without the sender being informed about the loss. This is acceptable for VoIP data.

The only control messages that are sent with UDP instead of TCP are for call handling so that an attacker can not deduce from the amount of TCP traffic if two peers are having a VoIP session or not.

To preserve anonymity, all UDP packets are of the same size (64 KiB) and there is always data sent, even if there isn't an active call. Thus, an attacker can not infer from the communication amount and bandwidth if there is an active call between two nodes.

3.1 Control message flow

When a node A wants to directly connect to another node B, it has to pass the following stages:

- Establish a simple TCP connection to B.
- Authenticate as onion node to avoid connecting to unrelated services running on the onion port.
- Do the onion handshake using OnionAuth module.

This all happens over TCP. Once the handshake has completed, the two peers can now exchange other control messages (see BLA) as well as user data (UDP), everything from this point on being encrypted with an ephemeral session key, so no one else can read their communication.

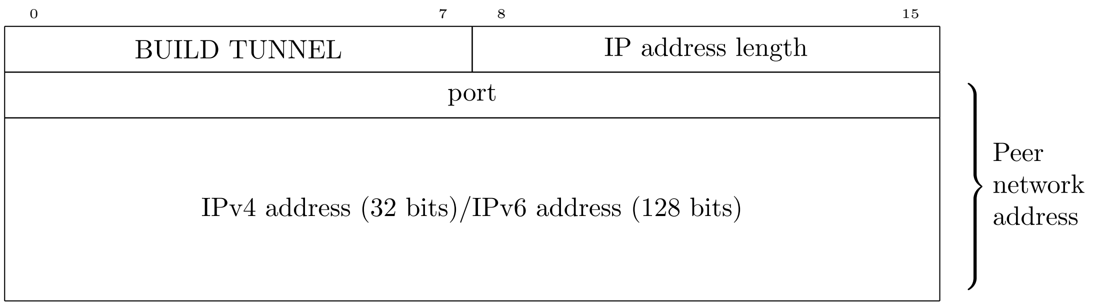
3.1.1 Control message types

- Authentication as onion node (TCP):

0 … 15	16 … 31
ONION MAGIC SEQUENCE	
version (1)	

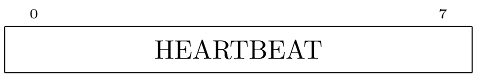
After establishing a TCP connection, the connection initiator sends this to the other node, which replies with the same message. This is for both to make sure that the communication partner is actually another onion node and not some different TCP service that coincidentally is running on this port. It also makes sure that both peers are running compatible versions (Only valid version at the time of this writing is 1).

- BUILD TUNNEL (TCP):



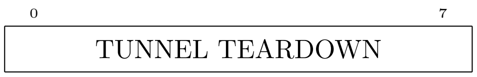
  Once A has established an encrypted connection to its first hop H1, it can send this message to it. H1 will then establish an unencrypted TCP connection to the hop H2 specified in this package and from that point on it will forward all TCP and UDP traffic it receives from A to H2 and for UDP packets also vice-versa. A can now do the authentication and handshake process with H2 over the encrypted connection to H1. It can iteratively add more hops like this until it reaches the desired target node.

- HEARTBEAT (TCP):



  This is sent at regular intervals from the tunnel initiator to the node at the end of the tunnel and vice-versa. If one of the two nodes does not receive a heartbeat in a certain time interval, it needs to assume that the tunnel is down and establish or wait for a new one.

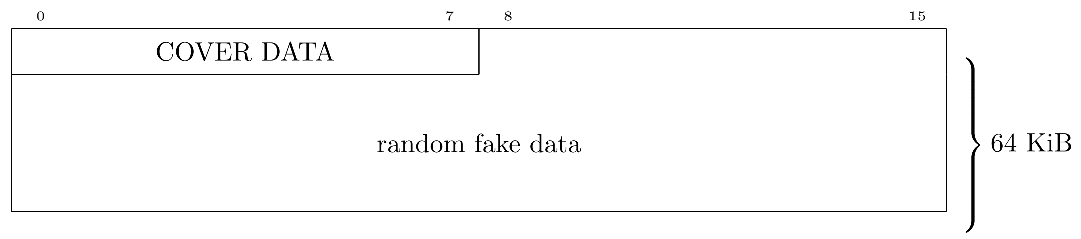
- TUNNEL TEARDOWN (TCP):



  This is used for the controlled destruction of a tunnel. The tunnel initiator has to send this to every hop, starting from the farthest to the closest one.

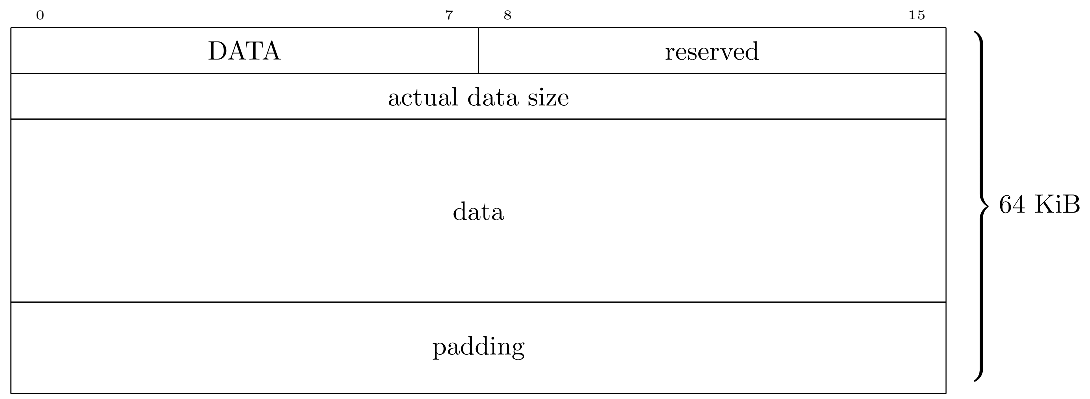
- COVER DATA (UDP):

  TODO Charlie ich weiss nicht warum die 64KiB-Klammer auf der rechten Seite so saubld nach unten verschoben ist. Bei DATA hab ichs genauso und da passt es.



Used for fake data if there is no active call between two peers. The packet always is of size 64 KiB. The recipient can just ignore the content.

- DATA (UDP):



Used for actual VoIP data. If a peer starts receiving these packets, it should start interpret it as a call request and start interpreting the data and responding with VoIP data once the user has accepted the call. Once the requesting peer starts receiving those answers, it knows that the call has been accepted.

3.2 Exception handling

TODO Charlie: Exceptions cause the app to blow up.