TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Informatics

Chair of Software Engineering

Security Analysis of Instant Messenger TorChat

Master's Thesis

Student: Rain Viigipuu
Student code: 072125

Supervisor: Alexander Norta, PhD

External Supervisor: Arnis Paršovs, MSc

TALLINN 2015

Declaration

I declare that this master thesis is the result of my own research except as cited in the references. The thesis has not been submitted before for any other degree or examination.

- - - - - - - - - - - - - - - - - - - - - - - -

(Date) (Signature)

Abstract

TorChat is a peer-to-peer instant messenger built on top of Tor network which not only provides authentication and end-to-end encryption, but also allows communication parties to stay anonymous and prevents third parties from even learning that communication is taking place.

The aim of this thesis is to document protocol used by TorChat and analyze security of TorChat and its reference implementation. The work shows that although TorChat design is sound, the implementation has several flaws which make TorChat users vulnerable to impersonation, communication confirmation and denial-of-service attacks.

This work is written in English and contains 53 pages of text, 68 chapters and 11 figures.

Annotatsioon

TorChat on võrdõiguslikus võrgus *(peer-to-peer)* töötav kiirsuhtlus vahend, mis on loodud kasutades Tor võrgu komponente ja mis ainult ei paku autentimist ja täielikku krüpteerimist, vaid lisaks sellele võimaldab ka omavahel suhtlevatel partneritel jääda täielikult anonüümseks ja vältida seda, et kolmandad osapooled ei saaks teada, et vestlus on üldse toimunud.

Selle magistritöö eesmärgiks on dokumenteerida TorChati protokoll ja analüüsida TorChati ja tema näidisrakenduse turvalisust. Töö tulemus näitab, et kuigi TorChati disain on korralik, leidub selle näidisrakenduses mitmeid vigu mis muudavad TorChat-i kasutajad haavatavaks identiteedi üle võtmise, suhtluse olemasolu tõestamise ja teenusest keeldumise rünnakutele.

See töö on kirjutatud inglise keeles ja sisaldab 56 lehekülge teksti, 68 peatükki ning 11 joonist.

Abbreviations

IP - Internet Protocol, a set of rules for sending data across a network

DNS - The Domain Name System, hierarchical distributed naming system for Internet

AES - Advanced Encryption Standard, a specification for the encryption of electronic data

RSA - the Rivest-Shamir-Adleman cryptosystem, a cryptosystem for public-key encryption

SHA-1 - Secure Hashing Algorithm 1

DER - Distinguished Encoding Rules to encode any data object into a binary file

ASN.1 - Abstract Syntax Notation One, standard for representing data in computer networking

PKCS.1 - Public-Key Cryptography Standards

TLS - Transport Layer Security, a cryptographic protocol for secure Internet communication

TCP - Transmission Control Protocol, one of the core protocols of the Internet protocol suite

UTF-8 - Universal Coded Character Set Transformation Format 8-bit

PGP - Pretty Good Privacy, data encryption and decryption program

GTK - cross-platform widget toolkit for creating graphical user interfaces

EFF - Electronic Frontier Foundation

IM - Instant messaging

DoS - Denial-of-Service

List of Figures

Contents

1 Introduction

Today secure communication over the Internet is a challenging task. Recently it became known that the architecture of the most popular instant messenger Skype has been redesigned to enable communication surveillance on its users [1]. U.S. government is using secret warrantless requests to obtain personal information stored by service providers [2] and is using legal means to obtain encryption keys from service providers [3]. Another aspect recently realized by society is that privacy cannot be achieved just by securing communication content. The communication metadata showing parties involved in the communication and their location might be even more sensitive information than the communication content and thus must be protected as well [4].

Therefore, in order to achieve communication privacy, a solution is needed which provides end-to-end encryption between communication parties and makes collection of metadata very hard.

TorChat [5] instant messenger is such a technology which is build on the top of Tor [6] and which not only provides end-to-end encryption between TorChat clients, but also hides location of TorChat users and prevents third parties from determining whom TorChat client is communicating with.

TorChat has been allegedly used to provide secure communication between doctor and his patients [7], and as recently revealed by court documents also to protect the location of the master mind behind the notorious marketplace Silk Road [8].

It is not simple to estimate the size of current TorChat user base. The study performed on February 4th, 2013, which used Tor hidden service address harvesting method described in [9], from the harvested 39,824 unique Tor hidden service onion addresses found 385 TorChat clients [10].

The security guarantees provided by TorChat have not been analyzed before and the only source for TorChat protocol description has been the source code of TorChat reference implementation.

The goal of this work is to describe protocol used by TorChat, provide security

analysis of the TorChat protocol and report on the vulnerabilities found in the audit of the latest TorChat original Python implementation version 0.9.9.553.

Note, that there are also other TorChat implementations available. TorChat2 [11] which is a Pidgin IM plugin, jTorChat [12] – TorChat implementation in Java, and Ruby-torchat [13] – TorChat implementation in Ruby. However, this work focuses only on the TorChat original Python implementation, which is available for MS Windows and is included in most Linux distributions and therefore is believed to have the largest user base. It serves also as a reference implementation of TorChat protocol, which means that all other implementations have been developed based on the Python implementation.

The rest of the thesis is organized in the following way. The next section provides introduction to Tor anonymity network and Tor hidden services. Section 3 gives overview of TorChat and its features from user's perspective. Section 4 documents protocol as used by TorChat. Section 5 defines methodology for analyzing security of instant messengers. Section 6 provides security analysis of TorChat instant messenger and its original Python implementation by answering the questions defined in Section 5. Finally, Section 7 provides summary of findings and Section 8 concludes the work.

2 Tor and Hidden Services

This section provides short description of Tor and Tor hidden services feature. The reader is welcome to skip over this section if he is familiar with Tor. The remaining part of the thesis assumes that the reader knows how Tor and its location hidden services work.

Tor is a software which allows people to keep their Internet activity private and anonymous. It also provides the tools and platform for developers who can create new applications with built in encryption, privacy and anonymity features. Those applications not only allow users to use the Internet services anonymously, but also allow them to provide different kind of Internet services like websites, instant messaging services or some other service while not compromising their privacy.

On the Internet privacy can be achieved partially by using the Internet services over encrypted connections. In this way only the data moving over the network is private. However the metadata which shows the source and the destination of the data and other properties which are not encrypted can still be monitored and analyzed. Tor network solves that problem by encrypting not only the data moving over the network, but metadata as well. This way it is not possible by interested parties (governments, intelligence agencies, companies, law enforcement, criminals etc.) to analyze the network traffic in order to find out user's behaviour and his interests.

To achieve that, Tor client creates private and secure pathway called Tor circuit through Tor network to the destination. The circuit is made out of three Tor nodes. The circuit is extended one hop at a time and each node only knows where the data came from (previous node) and where it should go (next node). There is no single node which knows all the nodes in the circuit. The client negotiates separate set of encryption keys for each hop in the circuit to make sure, that the connections cannot be traced.

When user wants to use regular Internet service anonymously, the connection has to exit from Tor network at some point. This happens through an exit node, which

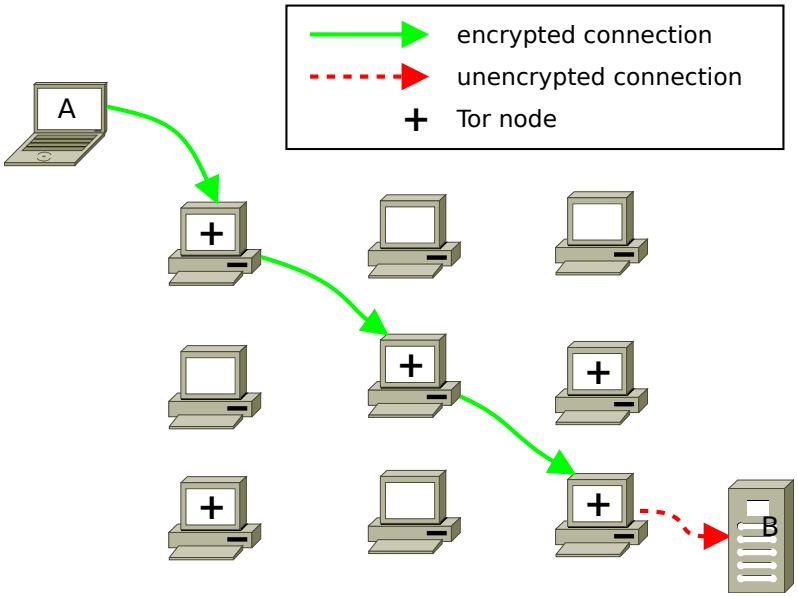

Figure 1: Network traffic moving through Tor network from A to B.

is the last node of Tor circuit. From that node forward the connection is encrypted only when the destination service supports encrypted communication. The way how traffic moves through Tor network is illustrated on Figure 1.

2.1 Hidden Services

Tor location hidden services is a feature of Tor that enables users to offer various services like websites or instant messaging server without revealing their location or identity. Other Tor users can connect to those services without knowing the service provider's network identity and without revealing their own. To achieve this goal Tor's rendezvous protocol is used [14].

When Tor client is configured to provide Hidden Service and is launched for the first time, it generates 1024-bit RSA key pair. From the public key it derives 16 character long hidden service address, also called as onion address.

In order to be reachable by other Tor users, hidden service has to advertise its existence in the Tor network. To do this the hidden service randomly picks some Tor relays, builds circuits to them and publishes in the public directory its hidden

service descriptor. The descriptor contains public key of the hidden service and list of introduction points where the service can be reached. Since there is a full Tor circuit established between the hidden service and the introduction point, the hidden service's IP address is hidden from the introduction point.

If someone wants to connect to the hidden service, it needs to know the onion address of the service. After the client has obtained the onion address of the service (by some out of band means), it can download the hidden service descriptor from the public directory. From the descriptor the client can extract hidden service's public key and list of introduction points through which the service can be reached. The client chooses one random Tor relay to be the rendezvous point for itself and the service, builds a Tor circuit to it and sends one time secret (rendezvous cookie) to it.

After the rendezvous point is set up the client puts together an introduce message containing the address of rendezvous point and one time secret. The client connects to one of the hidden service's introduction points via Tor circuit and sends the introduction message requesting it to be sent to the hidden service.

The hidden service receives the introduction message, decrypts it with it's private key and finds the information about the rendezvous point and the one time secret. Hidden service creates a circuit to the rendezvous point and sends a message with the on time secret.

The rendezvous point notifies the client, that the connection with hidden service is established and now the client and hidden service can use their circuits to the rendezvous point to communicate with each other. Rendezvous point simply relays the messages from the client to the hidden service and the other way around. Since the messages are end-to-end encrypted, the rendezvous point can't learn about the content of the messages.

2.1.1 Hidden service address

In Tor network, hidden services are identified and accessed by their so called onion addresses. An onion address is a 16 character hash with ".onion" suffix. The 16

character hash is computed as follows:

> The SHA-1 hash is calculated from hidden service's DER-encoded ASN.1 RSA public key (as specified in PKCS.1) [15]. Hash will be 160 bit long.
>
> The first half of the hash (80 bit long) is encoded to Base32, so it will only contain digits 2-7, letters a-z and will be exactly 16 characters long [16].

The reason why onion addresses are hashes and not human meaningful names is described in Zooko's Distnames essay. The essay argues, that a system giving out names in a network protocol has three desirable properties [17]: "human meaningful" – it means that they are highly memorable for human beings, "decentralized" – there is no centralized authority to hold the meaning of a name, and "secure" – there is only one, unique entity to which the name maps.

The essay states that a name system cannot have all three properties at the same time, but only two: decentralized and human meaningful (for example nicknames that people are choosing for themselves), secure and human meaningful (for example the current DNS system) and secure and decentralized (OpenPGP has those properties).

Onion addresses have been chosen to be secure and decentralized and therefore have the disadvantage of not being meaningful for humans. The reason for that kind of choice, is that while the onion addresses are secure, they are also self-authenticating. It means that when a client gets an onion addresses and requests the descriptor from Hidden Service Directory service, then it also receives the hidden service's public key and it can derive the onion address from it. It allows client to make sure that it is encrypting and sending the data to the right hidden service [16].

3 TorChat

TorChat is a peer to peer instant messaging solution built on top of Tor and its location hidden services feature. It can be used to interactively chat and exchange files between participants.

On a very high level view TorChat works by making every TorChat client available through Tor network as a hidden service. The hidden service's onion address is used as a unique identifier in TorChat. The onion address is a domain name allowing anyone to establish end-to-end encrypted and authenticated connection over the Tor network to the service behind that address.

TorChat is written in Python and uses wxPython library to draw GUI objects. To provide end-to-end encryption and anonymity features TorChat relies on the official Tor client software.

TorChat has been designed to be simple to use. For Windows version of TorChat everything required to run TorChat is included in TorChat Windows archive. On other platforms (Linux and Mac) TorChat requires to have Tor client and wxPython libraries installed. Besides that there is no configuration required – Tor hidden service is started by TorChat and keypair for the hidden service is generated when TorChat is launched for the first time.

However, the last update to the TorChat main code base was made on June 12, 2013. The executable packages have not been updated either. Because of this the latest TorChat Windows package available for download ships with out-dated Tor client which cannot be used anymore for connecting to the Tor network.

3.1 Managing Contacts and Conversations

TorChat's user interface is very simple and minimalist. The TorChat main window holds contact list and has only one button for changing the status (see Figure 2). The application menu is available through right mouse click somewhere inside the contact list window (see Figure 2).

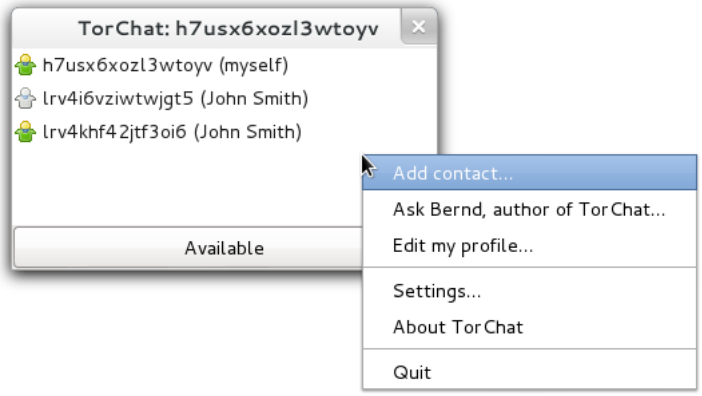

Figure 2: TorChat main window with application menu opened.

After TorChat is started for the first time it adds to the contact list user's own TorChat contact with profile name set to "myself" (see Figure 2). This allows for a user to easily find his TorChat identifier and see whether he is online, i.e., whether his onion address is reachable over the Tor network, since after the TorChat client is started, it may take a while until Tor circuits are established and hidden service descriptor published in the Tor directory.

To add a contact the "Add contact..." entry must be selected from the application menu (see Figure 2). The "Add new contact" window will open (see Figure 3) where the user can enter peer's TorChat identifier, profile name and introduction message which will be sent to the contact. If the profile name is omitted (the field is left empty), the profile name will be set to one received from the peer.

To remove a contact the "Delete contact..." entry must be selected from the application menu. After user confirms the deletion, the remote peer will be informed about deletion request and corresponding contact will be deleted from peers contact lists. Note, that if the peer is not online, the deletion request will not reach the peer and after peer will come back online the contact will be added back to the peer's contact list.

Contact's profile information can be seen when the user hovers mouse cursor over the contact's entry in contact list. The profile information shows always the latest profile name, profile text and avatar received from the contact (see Figure 4).

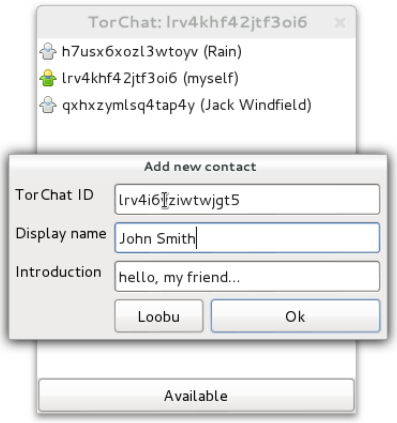

Figure 3: TorChat "Add contact" window.

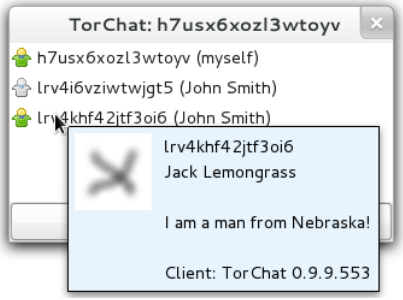

Figure 4: Contact's profile information box.

The icon in front of the contact indicates the status of the contact. If the icon is grey then the contact is offline. If the icon is blue ball/globe then TorChat is in the middle of handshake process with that contact. If the icon is green, then the contact is online. The icon can also be yellow and red, which means that the contact is online, but has set his availability status to away or long time away, respectively.

To start the conversation with a contact, a double click must be made on the contact name. This will open a conversation window with that contact (see Figure 5). The messages entered for the contact who is offline will be saved in `~/.torchat/<TorChat identifier>_offline.txt` file and will be sent to the peer when it comes back online. In the conversation window these messages will be marked with "[delayed]" prefix.

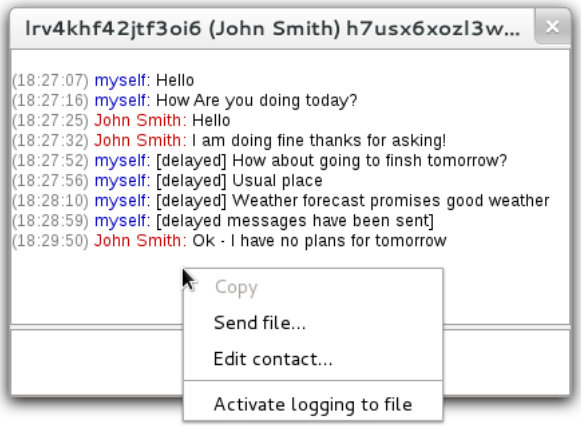

Figure 5: Conversation window.

The conversation window has its own menu which can be accessed via right click on the upper part of the window. The menu allows to send files to that contact, edit profile name shown in the contact list and turn on/off message logging (off by default). If the logging is enable the conversation with the peer is saved in the `~/.torchat/<TorChat identifier>.log` file.

To send a file the "Send file..." entry must be selected from the application menu. Files sent from the contacts are automatically saved as temporary files in `~/.torchat/torchat_incoming_<unique identifier>`. The user receiving the file can click on the "Cancel" button to stop the download and delete temporary file or "Save as..." button which will save the file and remove the temporary file (see Figure 6).

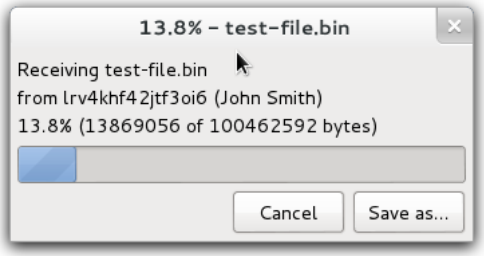

Figure 6: File transfer window (receiving in progress).

3.2 Configuration Options

TorChat can be configured using the "Settings" dialog of the application menu. The settings are stored in `~/.torchat/torchat.ini` configuration file, which can be edited also using a text editor.

TorChat's Settings dialog has three tabs: "Network", "User interface" and "Misc" (see Figure 7). Under the "Network" tab it is possible to change the network configuration stored in `torchat.ini`. There is no need to modify default network settings unless the user wants to run several TorChat clients in parallel [18].

The "User interface" tab can be used to set whether the TorChat contact window should be minimized when the application starts, whether the new windows should be opened automatically, the user interface language and whether the application should notify the user about new messages.

Under the "Misc" tab it is possible to configure whether the temporary files should be stored inside data directory (which is the folder from which the TorChat was executed), in operating system default location or user specified location.

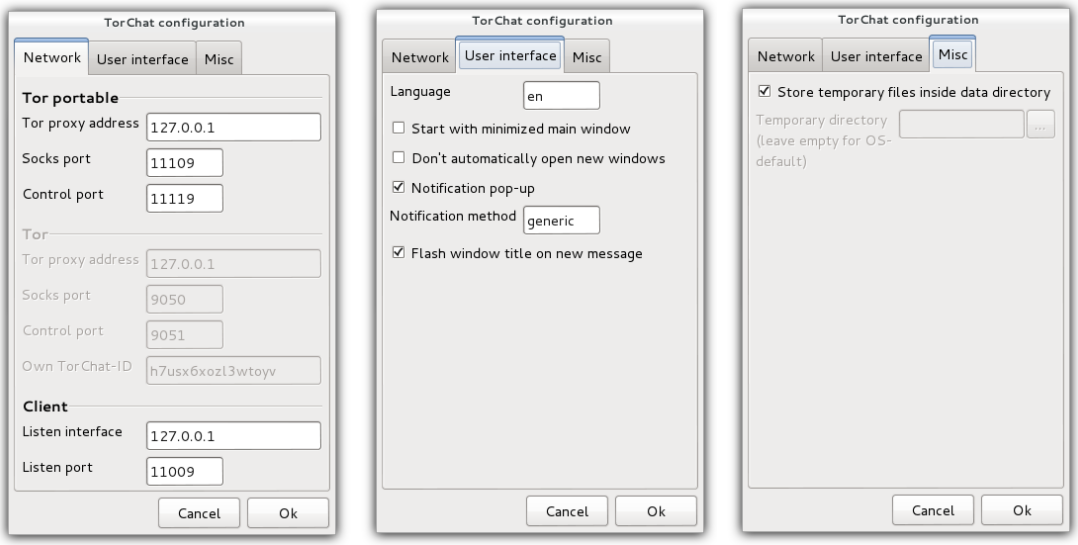

Figure 7: TorChat configuration window.

4 TorChat Protocol

This section describes protocol used by TorChat. The description provided here has been obtained by examining the source code.

4.1 Handshake

Before two parties can start to exchange messages they must perform handshake process. The goal of the handshake process is to establish trusted connections between parties, so after the process is completed, both parties know who they are communicating with, i.e., what Tor hidden service the other party controls.

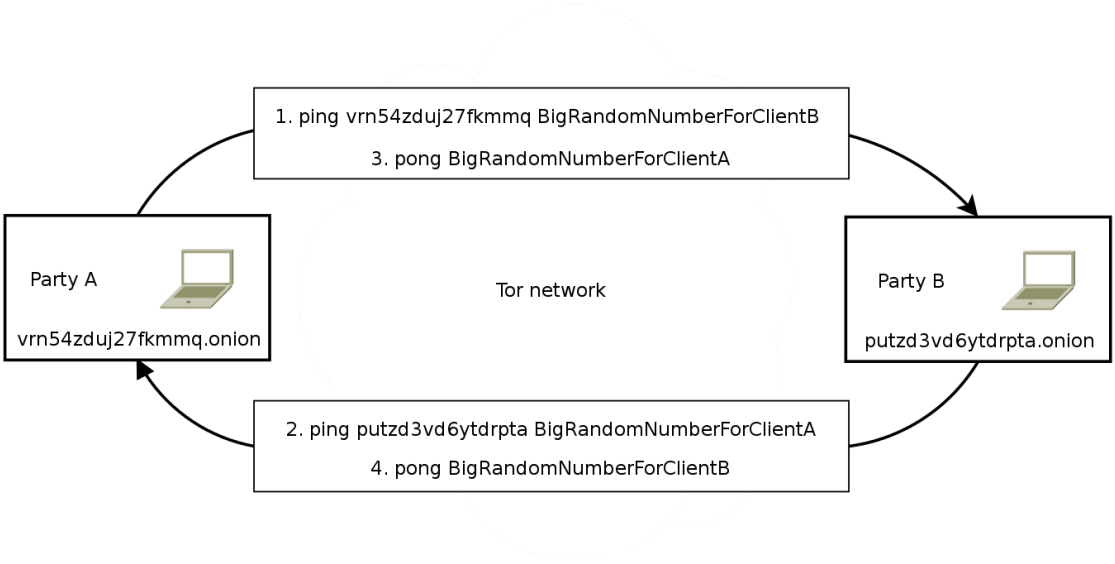

Figure 8: Handshake process.

Handshake process (see Figure 8) between TorChat clients party A and B:

1. Party A knows party B's onion address (established by out-of-band means). Party A establishes TCP connection over Tor network to party B onion address to port 11009. Party A sends `ping` message with its own onion address and a large random number. At this point party A knows with whom it has established secure outgoing connection. However, party B does not know who is behind the anonymous incoming connection.

2. Party B establishes TCP connection over Tor network to party A onion address received in step 1 to port 11009. Over the established connection party B sends `ping` message with its onion address and new random number generated by B.

3. Party B sends `pong` message with the random number received in step 1 to party A. At this point party B knows with whom the secure outgoing connection has been established and party A knows that the anonymous incoming connection belongs to party B, because only party B could have replied with the same random number as sent in step 1.

4. Party A takes parties B random number received in step 2 and sends it in the `pong` message over the outgoing connection established in step 1. At this point also the party B knows that anonymous incoming connection established in step 1 belongs to party A, since only the party A could have replied with the random number sent by party B in step 2.

After the handshake is completed, there are two end-to-end encrypted TCP connections between A and B and both A and B know the identity (onion address) of the other party.

4.2 File Transfers

File transfers are implemented using `file*` protocol messages. The files are split into blocks and sent using `filedata` protocol messages. To speed up a file transfer, several blocks can be sent before acknowledgement `filedata_ok` message is received. Block size used by TorChat is set to 8192 bytes and the number of blocks sent before receiving the acknowledgement is 16.

As an exception, all `file*` protocol messages are accepted on both – incoming and outgoing connections. However, TorChat sends only `filename` and `filedata` protocol messages on the incoming connection to avoid delaying of chat messages.

4.3 Protocol Messages

TorChat protocol messages are exchanged over TCP sockets. The messages are separated by a newline character '\n' (`0x0a` in hex). Whenever a full message (ending with a newline character) is received, it should be immediately processed.

Protocol message format is as follows:

`<command> <optional parameters> <UTF-8 encoded data>`

The `<command>` part may contain only lowercase characters a-z and underscore character '_'. Command and encoded data is separated by a space character (`0x20` in hex). Data part is UTF-8 encoded binary string.

Backslash and newline characters inside the data part must be escaped. Backslash character '\' (`0x5c` in hex) must be replaced with two characters "\/" (`0x5c` and `0x2f` in hex) and newline character '\n' (`0x0a` in hex) must be replaced by two characters "\n" (`0x5c` and `0x6e` in hex). The opposite replacement must be done after receiving a message.

In addition to command, the message might also contain parameters to the command. The parameters are separated between other parameters and data with a space character. Parameters cannot contain space and have the same rules for formatting as the command has.

The following subsections describe protocol messages, their purpose, when they can be sent and how they should be processed when received by the client application.

The words "must" and "should" are used interchangeably and do not have any special meaning.

4.3.1 not_implemented

If a peer receives unknown message, he should send `not_implemented` over the outgoing connection. If the outgoing connection does not exist, the connection on which the unknown message was received should be closed.

4.3.2 ping

This message should be sent immediately after establishing outgoing connection to a peer. It should contain peers own address without ".onion" suffix and 32 byte random cookie unique for the peer to which ping is sent.

Example: `ping vrn54zduj27fkmmq 13883080185381582480815986962179 4`

When a peer receives `ping` message on the incoming connection he should first check:

1. If the address in the `ping` message is 16 characters long and contains valid base32 encoding. If the address is not valid then the message should not be processed and the incoming connection must be closed.
2. If a repeated `ping` message contains different address than in the previous `ping` message received on the same connection then the message should not be processed and the incoming connection should be closed.
3. If a `ping` message contains address which is already assigned with another incoming connection for which the handshake process has been completed then the message should not be processed and the `not_implemented double connection` message should be sent on the incoming connection on which the `ping` message was received, but the connection should not be closed.
4. If the received `ping` message contains peer's own address, but cookie different from the one expected, the message should not be processed and the connection on which the message was received should be closed.
   Note that after a TorChat client is started it establishes connection to its own hidden service. If such bogus message is received when the TorChat has already established connection to itself, the case will be handled by "double connection" check above.

If the checks above succeed then a buddy with the peer's address specified in the `ping` message and status `STATUS_OFFLINE` should be created. Once the buddy is created, an outgoing connection should be established to the peer's address

specified in the `ping` message. Once the outgoing connection is established, the outgoing connection should be assigned to the buddy and buddy status should be updated to `STATUS_HANDSHAKE`. Over the outgoing connection established the peer should send the following messages:

1. his own `ping` message;
2. the `pong` message with the random cookie received in `ping` message;
3. `client` message;
4. `version` message;
5. `profile_name` and `profile_text` messages (if set);
6. `profile_avatar_alpha` and `profile_avatar` messages (if avatar is set);
7. `add_me` message (if the contact is in peer's buddy list);
8. `status` message.

If a repeated `ping` message is received, but buddy already has outgoing connection on which `pong` message has been sent then the `ping` message received should be ignored, unless `pong` reply from peer has not been received yet, in which case `ping` message should be sent over the outgoing connection.

4.3.3 pong

This message should be sent over outgoing connection as an answer to `ping` message received over incoming connection. The message must contain the random cookie received in `ping` message.

Example: `pong 13883080185381582480815986962179 4`

When a peer receives `pong` message on an incoming connection, he must process it as follows:

1. Extract random cookie from the message and find the first buddy who has been `ping`'ed with this random cookie. If a buddy with the random cookie cannot be found the `pong` message should be ignored.

2. If the buddy is found it must be checked whether the address assigned to buddy is the same address that has been specified in the `ping` message received over this incoming connection. If this check fails, the `pong` message should be ignored.

If these checks succeed the incoming connection should be assigned to the buddy and previous incoming connection (if it exists and is not the same) should be closed.

4.3.4 client

This message is sent in an answer to `ping` message (see Section 4.3.2). The message must contain the name of the client software.

Example: `client TorChat`

When received the peer should assign the received client software name to the contact.

4.3.5 version

This message is sent in an answer to `ping` message (see Section 4.3.2). The message must contain the version string of the client software.

Example: `version 0.9.9.553`

When received the peer should assign the received version string to the contact.

4.3.6 status

This message must be sent in an answer to `ping` message (see Section 4.3.2), immediately after the user has changed the status or profile information and in every 120 seconds. The message can contain three different status values: “available”, “away” and “xa” where “xa” stands for “extended away”.

Example: `status available`

When receiving this message, the client should update the contact's status information accordingly. If the `status` message is not received from a peer in 240 seconds the incoming connection should be closed.

4.3.7 profile_name

This message is sent in an answer to `ping` message (see Section 4.3.2) and to all peers whenever the user changes his profile information. The message is not sent if the user has not specified his profile name or the user changes his profile name to empty name.

Example: `profile_name John Smith`

If a client receives this message and the profile name for the peer is not set, the client should store the profile name in the contact list and show it next to peer's TorChat identifier in the contact window. If a client receives this message, but the profile name for the peer is already set, the profile name should not be changed, but the profile name received should be shown when a mouse is moved over the contact's entry in the contact window (see Section 3.1).

4.3.8 profile_text

This message is sent in an answer to `ping` message (see Section 4.3.2) and to all peers whenever the user changes his profile information. The message is not sent if the user has not specified his profile text or the user changes his profile text to empty text.

Example: `profile_text Business consultant with extensive experience`

The profile text received in this message should not be stored in the contact list, but should be shown when a mouse is moved over the contact's entry in the contact window (see Section 3.1).

4.3.9 profile_avatar_alpha

This message is sent in an answer to `ping` message (see Section 4.3.2) and to all peers whenever the user changes his avatar. The message is not sent if the user has not specified his avatar or the user removes his avatar.

This message contains uncompressed 64*64*8bit alpha channel. If there is no alpha channel in the avatar, the message has to be sent without the data part. This message has to be sent before sending `profile_avatar` message.

Example: `profile_avatar_alpha <alpha channel binary data>`

When receiving this message, the client should wait for peer's `profile_avatar` message to construct the avatar.

4.3.10 profile_avatar

This message is sent in an answer to `ping` message (see Section 4.3.2) and to all peers whenever the user changes his avatar. The message is not sent if the user has not specified his avatar or the user removes his avatar.

The message contains uncompressed 64*64*24bit image. This message must be sent after `profile_avatar_alpha` message is sent.

Example: `profile_avatar <image data as raw binary data>`

When receiving this message, the client should construct avatar taking into account alpha channel received in `profile_avatar_alpha` message and make peer's avatar visible when a mouse is moved over the contact's entry in the contact window (see Section 3.1).

4.3.11 add_me

This message is sent in an answer to `ping` message (see Section 4.3.2) if the peer is in the sender's contact list.

Example: `add_me`

4.3.12 remove_me

This message is sent to the peer when a user removes peer from the contact list. If the peer is not online, this message is not sent. After removing the peer from the contact list, peer's offline message queue file should also be wiped.

Example: `remove_me`

When receiving this message, the client should remove the peer from the contact lists and contact window, wipe peer's offline message queue file and close incoming and outgoing connection associated with the peer.

4.3.13 message

This message is sent to the peer whenever the user has entered text in the conversation window or after the peer comes online and there are unsent messages in the offline message queue.

The message may be sent only after `add_me` message has been sent or it is known that the peer is in the receiver's contact list.

Example: `message Hello, how are you?`

4.3.14 filename

The `filename` message is message initiating the file transfer.

The data part of the message contains information about the file being sent:

id – identifier generated by sender which uniquely identifies the file transfer.

file_size – size of the file in bytes.

block_size – specifies the chunk size that will be sent in `filedata` messages (the actual `filedata` messages can contain block in a different size).

file_name – specifies the name of the file as stored in the sender's system.

Every field is separated by a space character.

Example: `filename 2665323703 11 8192 testfile.txt`

4.3.15 filedata

This message is used to transport the actual data in a fixed size blocks. Every message contains the file transfer identifier, offset in the original file, lower case MD5 hash of the current data block and arbitrary size block of data itself. Each message should be answered with `filedata_ok` message after the receiver has successfully verified the hash of the data block. Hash of the block is calculated on unescaped data block.

The sender should send only a limited number of blocks ahead of incoming `filedata_ok` messages. For example, the sender should send the 5th block only after the 1st block's arrival is confirmed (`filedata_ok` message for the 1st block has arrived), the 6th only after the second block's arrival is confirmed and so on. The `filedata` messages must be sent in sequential order.

Example:
`filedata 2665323703 0 ee5a58024a155466b43bc559d953e018 line1\nline2`

4.3.16 filedata_ok

This message is sent as an answer to `filedata` message after receiver has successfully verified the `filedata` message. File sender can use this message to update the file transfer progress bar and to know that more blocks can be sent.

The message contains file transfer identifier and the offset position of the block that has been successfully received.

Example: `filedata_ok 2665323703 0`

4.3.17 filedata_error

The message is sent in case there have been some problem receiving the `filedata` message. The possible problem might be that the hash was wrong or the file offset was not the expected one (too much ahead that it should be, meaning that some blocks have been skipped or lost during temporary network outage).

The sender must respond to this message by restarting the file transfer at the offset specified in the message.

The message contains file transfer identifier and the file offset from which the file transfer should be resumed.

Example: `filedata_error 2665323703 0`

4.3.18 file_stop_sending

The message is sent to the file sender if the receiver has canceled the file transfer. After this message is received, the sender should stop sending the file. The user must be notified that the file receiver has canceled the file transfer.

Example: `file_stop_sending 2665323703`

4.3.19 file_stop_receiving

The message is sent to the file receiver if the sender has canceled the file transfer or the sender has received from the receiver `file_stop_sending` message. After this message is received, the receiver should not expect any more messages related to file transfer identified. All allocated resources should be freed, temporary files should be wiped and user should be notified about the cancellation.

Example: `file_stop_receiving 2665323703`

5 Analysis Methodology

Security analysis methodology presented in this section is based on EFF's "Secure Messaging Scorecard" [19], but extend with additional criteria which concern not only the privacy of message contents but also the privacy of communication metadata and other issues related to secure and anonymous use of instant messaging in practice.

5.1 Is the communication protected in transit?

This criteria considers an attacker who controls network connection between the user and IM service provider. This question should answer how the data is protected from such attacker. What the attacker can learn by observing the connection? Can he learn whether the user is using IM service? Can he learn who the user is communicating with? Can he use his powers to tamper with the communication line to execute some attack, other than the denial-of-service attack?

5.2 Is the communication protected from abuse by the provider?

This criteria considers malicious IM service provider whose objective is to compromise user's privacy, communication integrity or even user's device on which the IM software is running. The question should give an answer what the user is trusting to the service provider.

5.3 Can someone impersonate user's identity?

This criteria should answer whether the user has means to verify cryptographic binding between the communication channel and his contact's identity or he has to trust IM service provider on that.

5.4 Are past communications secure if user's keys are stolen?

This criteria considers an attacker who has collected encrypted communication on any point between the user and the intended recipient and then obtains long-term encryption key from the user in order to decrypt collected past communication. This question should answer whether the forward secrecy is used, i.e., whether the user's long-term asymmetric key is used in signing mode only to establish short-term encryption key?

5.5 Is the source code available, crypto design well-documented, open to independent review?

This criteria considers whether the source code of the software is freely available and whether the design of the software is well documented.

5.6 Can the service be used anonymously?

Can the user register IM service account anonymously? Is the user's connection IP address concealed from his contacts and from the IM service provider?

5.7 Who has access to the user's profile information?

This criteria considers how the user's provided profile information is protected from third parties. Is this information available only to the persons in the contact list, to IM service provider or someone else?

5.8 Who has access to the user's presence information?

This criteria considers both – the presence information (reveals whether the user is logged into the IM service) and availability information (reveals user's specified status in IM service, i.e., busy, away, available). Is this information available only to the persons in the contact list, to IM service provider or someone else?

5.9 Who has access to the user's contacts information?

This criteria should answer who except the parties involved in the communication can learn about these parties being in contact.

5.10 Is the user protected from denial-of-service attacks?

The question to be answered here is whether the IM service is vulnerable to any attack which could be used by an attacker to deny user's access to the service.

This includes any kind of targeted attacks which will result in messaging client crash, intensive CPU or memory usage or in any other way will create trouble for user to use the service.

5.11 What forensic evidence the software leaves on the user's device?

This criteria should answer what kind of IM related information can be extracted from the device in case the device is obtained by third party. Things to look for — contact list, messaging history, logs, keys, file transfer history, offline message queue, etc.

5.12 Is the software available from trusted source and can its integrity be verified?

This criteria should answer to the following questions:

1. How the software is made available? Are the distribution methods and channels trusted?
2. If the binary packages are provided, are there ways to reproduce them?
3. Are there ways available to verify package authenticity?

6 Security Analysis

This section gives detailed answers to the questions defined in the previous section.

6.1 Is the communication protected in transit?

TorChat does not implement its own encryption, but fully relies on confidentiality and authenticity guarantees provided by Tor and Tor hidden service design.

Adversary eavesdropping on communication channel between the TorChat client and Tor network would see encrypted traffic protected by 4 layers of encryption, where the innermost layer contains end-to-end encrypted TorChat protocol data exchanged between TorChat peers. Every layer is protected by 128-bit AES key negotiated using 1024-bit RSA ephemeral DH keys [15].

Therefore, the only information the attacker can learn is that the user is using Tor. This fact is easy to learn since entry nodes of the Tor network can be identified by their IP addresses.

6.2 Is the communication protected from abuse by the provider?

In the TorChat context we can consider the Tor network being the IM service provider.

There are several entities involved in establishing connection between two Tor hidden services (TorChat peers).

If the hidden service directory where TorChat's hidden service descriptor is published is under adversary control, the adversary can only learn the same information as these who are requesting that descriptor. This includes introduction points which can be used to contact the hidden service, public key (onion address) of the hidden service and the time when the descriptor was published.

An adversary operating node which acts as rendezvous point between TorChat peers would still have to break the final layer of end-to-end encryption between TorChat peers. The attack by rendezvous point is complicated even further, because rendezvous points are chosen randomly by Tor client, both end-points from the perspective of rendezvous point are anonymous, and most likely a single rendezvous point would route only one direction of TorChat communication.

To sum up, if the security assumptions implied by Tor hold, the IM service provider is not capable of attacking TorChat users.

6.3 Can someone impersonate user's identity?

The following subsections contain discussion on several attacks that can be used to impersonate TorChat user.

6.3.1 Impersonating Tor hidden service

For authenticity guarantees TorChat exploits the self-authenticating nature of onion addresses, where the onion address represents public key of hidden service long-term 1024-bit RSA key.

As can be seen in the description of handshake process (see Section 4.1), after the handshake is completed both TorChat peers are sure that the opposite party is in control of the private key corresponding to the onion address.

In order to impersonate TorChat peer an adversary would have to compromise hidden service 1024-bit RSA key. The RSA keypair is generated by Tor on the first use of TorChat. In Unix systems the generated key is stored in the file `~/.torchat/Tor/hidden_service/private_key`, which is readable only by the user.

While the security of 1024-bit RSA today is considered weak [20], there are not know any cases where 1024-bit RSA key would have been compromised. Since this long-term key is used only for authentication, the use of 1024-bit RSA for the present is tolerable.

Alternatively, since the onion address is base32 encoded first 80-bits of SHA-1 digest of RSA public key, an adversary may try to find different RSA key which would collide to the same onion address. However, the complexity of such attack is estimated 2^{80}, which is nearly the same as factoring 1024-bit RSA key. In case the attacker can find a different RSA key which produces the same onion address, there would be a race condition with the legitimate user's published hidden service descriptor [21].

6.3.2 Spoofing `pong` message

As can be seen from the TorChat protocol description (see Section 4.1) a peer authenticates incoming connection by checking if the random number received in `pong` message matches the random number that was sent in `ping` message over the outgoing connection. Thus, if the attacker is able to guess the random number which was sent in the `ping` message, he can spoof `pong` message thereby impersonating incoming connection.

In TorChat implementation the random number sent in `ping` message is generated by calling `random.getrandombits(256)` method which will return integer from 0 to $2^{256} - 1$ generated by MersenneTwister pseudo random number generator. The Python manual states that it should not be used for security purposes and suggests to use cryptographically secure pseudo-random number `os.urandom()` or `SystemRandom` [22] instead.

Even if the attacker can predict random number used in `ping` message, the impact of the attack is limited, since the attack has to be executed at the time when vulnerable victim performs handshake and before the legitimate peer has answered with his `pong` message. Furthermore, in case of successful impersonation the attacker can only send impersonated messages, but not read the responses. The exceptions are file transfers which are sent over the incoming connection (see Section 4.2). Thus, if the attacker could convince the victim to send a file, the file would be received by the attacker over victims incoming connection.

6.3.3 Impersonation at GUI level

Since in the TorChat peer identities are onion addresses which are hard to memorize, the TorChat provides possibility to assign arbitrary name to the contact which will be shown next to TorChat identifier in the contact list. However, the way it is implemented in the TorChat opens contact list confusion attacks which may lead to a successful impersonation attack.

As described in Section 3.1, when adding a new contact, It is possible to specify contact's profile name/ If the name is left empty then it is set to the name which is specified in the `profile_name` message received from the contact. Although, if the name is set either manually or by receiving `profile_name` message, it will not be overwritten.

This allows for the remote peer to set profile name specified by him if the user does not specify one when adding the contact. This is not a problem since adding the contact is an operation consciously performed by the user and the user has opportunity to set the profile name. However, if someone adds user to his contact list that someone is added to the user's contact list automatically and contact's profile name is set to the name sent by that someone in `profile_name` message. The user has no way to control what gets added to his contact list.

Thus if the attacker knows whom the victim is chatting with, the attacker can generate similarly looking onion address [23] and set the same profile name and add victim to the contact list. This way the victim will have several contacts with the same contact name and similar TorChat identifier which can be used by the attacker to start the conversation and trick the user into thinking that the window with attackers conversation is the intended TorChat peer (see Figure 9).

Figure 9: Two similarly looking TorChat contacts. The second one has been added automatically by the attacker.

The hint which can be used by the victim to determine the impersonated TorChat identity is to look at the order of contacts in contact list. The contacts in the contact list are ordered by adding newly added or edited contact to the bottom of the list.

This kind of attack would be prevented if TorChat would ask for confirmation before contact is added to the contact list. Note, that arbitrary contacts can be added to the contact list also by exploiting the contact list manipulation flaw described further in Section 6.10.3.

6.4 Are past communications secure if user's keys are stolen?

According to the chapter "1.1 Keys and names" in Tor specification [24] there a 3 kinds of keys used in Tor:

> A long term "Identity key" used only for signing. In TorChat this is the key that is used to identify the Hidden Service (TorChat user).
>
> A medium term "Onion key" which is used to decrypt onion skins when the circuit is in the making. This key is rotated once a week.
>
> A short term "Connection key", used to negotiate TLS connections. Those keys can be rotated as often as they like but at least once a day.

From chapter "2. Connections" [25] in Tor specification it is written that all the connections between Tor relays or between Tor client and Tor relay use TLS/SSLv3 for link authentication and encryption. If the adversary has collected some encrypted network traffic, then it is encrypted with short term "Connection key(s)" which are rotated at least once day. So even if the long term "Identity key" is compromised, it cannot be used to decrypt the collected network traffic.

6.5 Is the source code available, crypto design well-documented, open to independent review?

Both TorChat and Tor source code is freely available and open to independent review. Regarding the documentation situation is different.

Tor has high level overview of main features and principles published in Tor's website. The code repository contains specifications which describe in depth the protocol and how Tor network operates.

TorChat, on the other hand, doesn't have much documentation available. There is some general information available on its GitHub page and there is also some general documentation regarding the configuration and different working modes available in the "docs" folder together with source code. The protocol of the TorChat is not described in any separate document.

6.6 Can the service be used anonymously?

The anonymity of person using TorChat can be reduced to anonymity guarantees provided by Tor and hidden services design. In literature there have been several attacks described which could be used to locate Tor hidden service or Tor user [9]. The most popular being traffic confirmation attacks.

The subsections below discusses some anonymity aspects which affect specifically TorChat.

6.6.1 Deanonymization by a malicious guard node

One of the weaknesses of Tor and other low-latency anonymity networks is that if the attacker can monitor both ends of the communication channel, then he can correlate the data volume and timing information and this way compromise the anonymity. In Tor network it means that attacker needs to control the circuit's first and last relay. In case of hidden services it means that attacker needs to

control just the entry node which is chosen by the hidden service, since the other end is already under the control of attacker.

If Tor client would always choose new entry guard for each circuit the Tor would eventually pick attackers controlled entry node and user's privacy would be compromised. To protect against this, Tor client is randomly choosing set of fixed relays which will act as entry guards. Guard node rotation time is set by the configuration parameter `"GuardLifeTime"` and by default it is 60 days [26].

In the configuration file that is shipped with TorChat (`~/.torchat/Tor/torrc.txt`) the number of entry guards is set to 6 (`NumEntryGuards 6`). Considering, that Tor's default value is coming from directory authority and is usually 1 or 2 or if the number isn't found in consensus file, it defaults to 3 [27] then TorChat's use of 6 nodes compared to Tor's default setting increase the risk of TorChat user deanonymization by half and therefore should be reconsidered.

6.6.2 Deanonymization by message contents

It is well known fact that Tor provides anonymity only on the transport protocol level. If the user in his TorChat profile or messages sent exposes information which can be used to identify him, the Tor anonymity guarantees will not help.

The TorChat users have to be especially careful with the links that are sent to them in the conversation. If the user uses Tor only for TorChat and other applications go directly to the Internet, the user by opening link from the TorChat window can expose his real IP address to the server hosting the web site, which if under the control of attacker, will allow attacker to find out TorChat user's real IP address. To make it easier to slip TorChat makes all the links clickable (see Figure 10).

This is the simplest way to deanonymize TorChat user. It is not recommended to remove link clickability, since the user will most likely copy and paste the same URL in browser. However, it is recommended that when the user clicks on the link, the warning message is displayed which warns about deanonymization threat, with the option to click checkbox to not warn again.

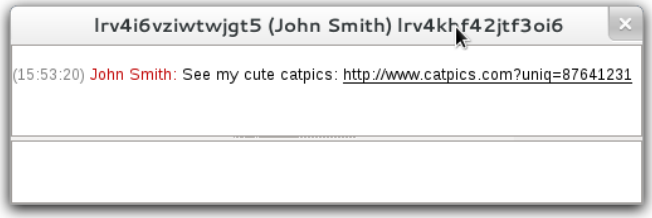

Figure 10: Clickable links in TorChat.

6.7 Who has access to the user's profile information?

It would be expected that user's profile information (profile name, description and avatar) would be available only to contacts in the contact list.

However, there is no contact authorization in TorChat. Anyone who knows user's TorChat identifier can add that address to TorChat contact list and after handshake is performed successfully that "anyone" will be added to the victims contact list and victim's TorChat client will send profile information without authorization to that "anyone". Even more, that "anyone" does not even have to perform successful handshake. The profile information will be disclosed right after receiving that "anyone's" `ping` message.

The correct behaviour would be for TorChat to ask authorization from TorChat user before contact is added to the contact list and profile information is disclosed.

6.8 Who has access to the user's presence information?

It would be desirable that user's presence information is disclosed only to contacts in the user's contact list. As described in the previous section, TorChat discloses profile information including availability information to anyone who establish TCP connection to TorChat.

However, even if TorChat would disclose availability only to contacts in the contact list, the presence information is available to every client in Tor network who is

able to download hidden service descriptor. The descriptor is updated in directory server once an hour or whenever its content changes [28].

By observing those times the adversary can build activity graph and use it to correlate TorChat activity with other activities thereby disclosing the real identity of the TorChat user.

There is no solution for that and the only recommendation would be to make sure that TorChat is on all the time.

Interesting to note, that someone who is in the TorChat user's contact list can distinguish between network outage and TorChat client restart, since the random number used to authenticate the peer is not regenerated if the connection between peer tears down.

6.9 Who has access to the user's contact information?

Thanks to the Tor and hidden service design, the TorChat peers communicating should be the only parties that know about communication taking place and for anyone else finding who communicates with whom should be very difficult.

However, there is basic communication confirmation attack possible due to the way how TorChat protocol handles `ping` message received which contains TorChat identifier from a peer with whom the handshake has already been established.

As can be seen in the protocol description (see Section 4.3.2), if the `ping` message is received with a peer identifier with whom the handshake has already been established, the TorChat will ignore the `ping` message and will send `not_implemented double connection` on the connection on which `ping` message has been received. This can be used by an adversary test if the victim has established handshake with some other TorChat peer.

The simple fix would seem to just ignore the double `ping` message without informing the sender about the double connection. However, the proper fix would be to handle double `ping` message in a way which prevents attacker to distinguish from the case when double `ping` message is ignored and the case when TorChat

tries to establish the back-connection to the address specified in the spoofed `ping` message.

6.10 Is the user protected from denial-of-service attacks?

Since TorChat relies on Tor hidden service design, any denial-of-service attack against Tor hidden services is applicable also to TorChat. To make TorChat client unavailable one can use general attacks against Tor hidden services described in [9].

The subsections below focuses on application level denial-of-service attacks which apply specially to TorChat.

6.10.1 Memory exhaustion through network read

TorChat does not limit the length of any message and buffers bytes into the memory until command separator (newline character) is received. This can be used trivially by the attacker to exhaust available memory on victims system by sending endless stream of data over the victim's outgoing connection (since victim's incoming connection expects to receive keep alive `status` message every 120 seconds). How fast the memory will be exhausted on the victim's system depends on victim's system memory size and the speed in which victim can receive data over the Tor network.

6.10.2 Memory exhaustion through chat message

More efficient but also less invisible memory exhaustion attack can be achieved by sending large chat message in `message` command. For instance, 20MB large `message` command displayed in chat window will cause TorChat process to consume around 1GB of memory.

6.10.3 Attacking via `profile_name` message

As described above, TorChat does not enforce size limit for messages received. This allows an attacker to send arbitrary large profile name. Large profile name will cause TorChat GUI to hang while GTK tries to update contact list window. This will also prevent victim from removing the attacker from the contact list since the profile name is shown in "confirm deletion" window, which will not be created by GTK if wider or taller than 32767 pixels. Victim can remove the attacker by first deleting profile name in "Edit contact" menu.

TorChat also fails to validate the profile name before writing it into the contact list (`buddy-list.txt`). The file stores every contact in a separate line. The line starts with contact's TorChat identifier (onion address) and follows with contact's profile name separated from TorChat identifier with a space character.

The lack of validation allows to add arbitrary lines into the contact list by sending profile name which contains escaped newline characters.

The contact list is saved right after the `profile_name` message is received, however, injected lines will be read only after TorChat is restarted. The injected lines will be lost if some contact is added or removed from the contact list before TorChat is restarted.

This can be exploited by an attacker to cause effective DoS attack by adding thousands of contacts in the contact list. On start-up TorChat will try to create and establish connection to all the contacts in the contact list which will cause large Tor activity and will prevent TorChat GUI from starting. The only solution for the victim to be able to use TorChat again is to manually clean `buddy-list.txt`.

6.10.4 Attacking via multiple `add_me` messages

TorChat does not ask for user's consent before peer's request to be added to the contact list is processed. The attacker can exploit this by flooding the user with dummy contacts and messages.

There has been a discussion about adding a block list feature to the TorChat [29],

however, no solution has been implemented. The blacklist approach might not be effective since the attacker can introduce new TorChat identity without a significant cost. More appropriate solution would the whitelist approach were user is asked for confirmation before contact is added to his contact list.

6.10.5 Attacking vie multiple `filename` messages

Peers being in the contact list can send files which will be automatically accepted by the TorChat and download will start in a separate window (see Section 3.1).

The attacker can initiate many file transfers by sending many `filename` messages. This will fill up victim's screen with file transfer windows and cause memory on the victim's machine to be exhausted very fast resulting in a very efficient denial-of-service attack (see Figure 11).

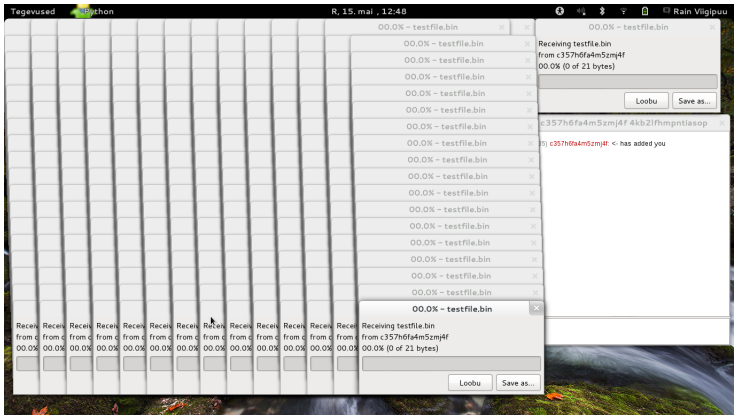

Figure 11: Denial-of-service attack by multiple file transfer windows.

The impact of the attack could be significantly reduced if the file transfers would be shown in a single file transfer window and if arbitrary contacts would not be added to the contact list without the user's consent.

6.11 What forensic evidence the software leaves on the user's device?

All user-specific TorChat related files are stored in `~/.torchat/` directory. This includes `buddy-list.txt` containing user's contacts, `torchat.ini` containing TorChat configuration parameters including user's profile name and profile text, `avatar.png` (if set by the user), logged conversations in `<TorChat identifier>.log` files (by default logging is not enabled), offline message queue in files of form `<TorChat identifier>_offline.txt`, and temporary files received from other peers in form `torchat_incoming_<unique identifier>`.

The `~/.torchat/Tor/` directory stores Tor hidden service long-term private key in the `hidden_service/private_key` file, Tor configuration in `torrc.txt` file and Tor cache in `tor_data/` directory.

TorChat uses secure erasing (wiping) in several occasions. Received temporary files are wiped as soon as user saves the file or cancels the download. The offline message queue files are wiped as soon as messages have been delivered or contact has been removed from the contact list.

6.12 Is the software available from trusted source and can its integrity be verified?

It is important to obtain the TorChat application from a trusted source to be sure about its integrity. The original source of TorChat is available in the TorChat's project page on GitHub [5]. Since GitHub is available over secure connection and all version history is preserved in GitHub, it is safe to assume that TorChat obtained from the GitHub has not been modified by some adversary.

The GitHub's download section [30] contains prebuilt packages of TorChat releases available for MS Windows operating system and Debian Linux distribution.

6.12.1 Linux

TorChat is available also in several Linux distribution package repositories:

Debian Linux stable version 8 has the latest version of TorChat 0.9.9.553 [31].

Ubuntu Linux version 14.04 [32] and the latest version 15.04 [33] both have TorChat version 0.9.9.553.

Arch Linux has TorChat 0.9.9.553 available in their Arch User Repository (AUR) [34].

Both Debian and Ubuntu apply minor patches to the versions they are providing through package repository: Spanish translation of TorChat, change the way how SOCKS proxy Python module is found in the system and update the wxWidgets Python library version upgrade from 2.8 to 3.0. Only Ubuntu 14.04 is missing the patch to upgrade wxWidgets Python module.

Arch Linux does not apply any patches and provides unmodified TorChat version.

6.12.2 Windows

For MS Windows users TorChat GitHub page provides executable which contains everything needed to run TorChat out of box – including the Tor client. The Windows executable allegedly [35] has been compiled using PyInstaller [36].

However, the latest windows executable ships with outdated Tor client version 0.2.2.39 [37]. This Tor version has several vulnerabilities including the heartbleed, which allows a malicious Tor guard node [38] to compromise TorChat's long-term private key. Fortunately, the version of Tor is too outdated to even connect to the Tor network [39].

7 Summary of Findings

Here is the list of security issues found ordered by their significance.

1. TorChat processes contact requests and updates contact list without asking for user's consent. This allows an attacker to harvest profile (Section 6.7) and availability (Section 6.8) information, allows to execute contact confusion and impersonation attacks at GUI level (Section 6.3.3) and makes denial-of-service attacks (Section 6.10.2, 6.10.3, 6.10.4, 6.10.5) easier to execute.

2. Due to the way how TorChat handshaking process is implemented it is possible for an attacker to verify if two TorChat clients who are online have established TorChat handshake thus allowing an attacker to execute communication confirmation attacks (Section 6.9).

3. TorChat does not enforce length limit for the received protocol messages and their parts. This allows an attacker to execute efficient denial-of-service attacks against TorChat client (Section 6.10.1, 6.10.2, 6.10.3).

4. TorChat fails to validate contents of received `profile_name` message before writing it into the contact file. This can be exploited by an attacker to add arbitrary contacts to the victims contact list and can be used to execute permanent denial-of-service attack which will result in victim's TorChat client failing to start (Section 6.10.3).

5. TorChat uses cryptographically insecure pseudo-random number generator to generate random numbers used in TorChat handshaking process. This may allow an attacker to impersonate incoming connection of victim's contact thus being able to send messages on behalf of that contact and receive file transfers designated for that contact (Section 6.3.2).

6. TorChat runs Tor with non-default parameters which makes TorChat users compared to other Tor users being more easier to deanonymize by a malicious Tor guard node (Section 6.6.1).

7. TorChat makes links automatically clickable without warning the user about possible deanonymization attacks (Section 6.6.2).

8 Conclusions

The objectives set in the introduction were achieved. TorChat protocol has been documented, reference implementation audited and as a result of security analysis several security considerations were revealed which needs to be considered when using TorChat.

The designer of TorChat has made several smart design choices by exploiting Tor hidden service's self-authenticating nature to provide authentication between TorChat peers and by leaving all the encryption and anonymity part to be handled by well tested and widely used Tor software.

However, several implementation flaws were found in the TorChat implementation which open TorChat users to denial-of-service attacks and prevent TorChat from achieving privacy guarantees it could provide in theory.

Despite the flaws found, the use of TorChat might still be secure in scenario where peer's onion address does not became known to an adversary interested in attacking the person behind the TorChat address.

Fortunately, the fixes for the vulnerabilities found can be relatively easily implemented in the code without requiring to change the design of TorChat.

Kokkuvõte

Sissejuhatuses püstitatud töö eesmärgid on täidetud. TorChat-i protokoll on dokumenteeritud, näidisrakenduse audit läbi viidud ja turvaanalüüsi tulemusena leitud asjaolud, mida tuleb arvestada TorChat-i kasutades, selgitatud.

TorChat-i looja on teinud mitmeid tarku otsuseid rakendust disainides otsustades ära kasutada Tor-i peidetud teenuste omadust olla iseennast autentivad ning kasutada seda TorChat-i klientide omavaheliseks autentimiseks. Samuti on mõistlik otsus jätta kõik krüptograafia ja anonüümsuse tagamisega seotu laialt kasutuses oleva ning põhjalikult testitud Tor tarkvara hooleks.

Siiski on TorChat-i näidisrakenduses mitmeid realisatsioonist tulenevaid puudusi mis võimaldavad TorChat-i kasutajate vastu teha teenuse kasutamist takistavaid ründeid ning ei täida täielikult TorChat-i lubadust privaatsusele, kuigi rakendus seda teoorias võimaldab.

Kuid hoolimata leitud puudustest võib TorChat-i kasutamine olla endiselt turvaline olukorras, kus kasutaja aaadress ei ole potentsiaalsele ründajale teada.

Õnneks on leitud puudused suhteliselt lihtsalt kõrvaldatavad rakenduse koodi parandades ning ei vaja TorChat-i rakenduse disaini muutmist.

References

[1] Ryan Gallagher. Timeline: How the World Was Misled About Government Skype Eavesdropping. July 2013. http://www.slate.com/blogs/future_tense/2013/07/12/skype_surveillance_a_timeline_of_public_claims_and_private_government_dealings.html.

[2] James Risen and Nick Wingfield. Web's Reach Binds N.S.A. and Silicon Valley Leaders. June 2013. http://www.nytimes.com/2013/06/20/technology/silicon-valley-and-spy-agency-bound-by-strengthening-web.html.

[3] Glenn Greenwald, Ewen MacAskill, Laura Poitras, Spencer Ackerman, and Dominic Rushe. Microsoft handed the NSA access to encrypted messages. July 2013. http://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data.

[4] Privacy International. What is metadata? https://www.privacyinternational.org/?q=node/53 (last visited 16.05.2015).

[5] Bernd Kreuss (author of TorChat). TorChat, January 2014. https://github.com/prof7bit/TorChat.

[6] The Tor Project, Inc. Tor: Overview. https://www.torproject.org/about/overview.html.en (last visited 16.05.2015).

[7] Bernd Kreuss. Interview with Bernd Kreuss of TorChat, October 2013. http://www.reddit.com/r/onions/comments/1l9k8m/interview_with_bernd_kreuss_of_torchat/ccnmivi.

[8] Joe Mullin. Silk Road trial: FBI reveals what's on Ross Ulbrichts computer, January 2015. http://arstechnica.com/tech-policy/2015/01/silk-road-trial-fbi-reveals-whats-on-ross-ulbrichts-computer/.

[9] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *IEEE Symposium on Security and Privacy'13*, pages 80–94, 2013.

[10] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Content and popularity analysis of Tor hidden services. *CoRR*, abs/1308.6768, 2013.

[11] Bernd Kreuss (author of TorChat). Torchat2 readme. `https://github.com/prof7bit/TorChat/blob/torchat2/README.markdown`.

[12] jTorchat, October 2014. `https://github.com/jtorchat/jtorchat`.

[13] torchat - the Ruby implementation, August 2012. `https://github.com/meh/ruby-torchat`.

[14] The Tor Project, Inc. Tor: Hidden Service Protocol. `https://www.torproject.org/docs/hidden-services.html.en` (last visited 16.05.2015).

[15] The Tor Project, Inc. Tor Protocol Specification, 0.3. Ciphers, Feb 2015. `https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt#n71`.

[16] Tor Project Wiki. Hidden Service Names. 2013. `https://trac.torproject.org/projects/tor/wiki/doc/HiddenServiceNames`.

[17] Wikipedia. Zooko's triangle. `http://en.wikipedia.org/wiki/Zooko%27s_triangle` (last visited 16.05.2015).

[18] Howto: Run more than one instance of TorChat, May 2008. `https://github.com/prof7bit/TorChat/blob/torchat_py/torchat/doc/howto_second_instance.html`.

[19] Electronic Frontier Foundation. Secure Massaging Scorecard, January 2015. `https://www.eff.org/secure-messaging-scorecard`.

[20] Elaine Barker and Allen Roginsky. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. January 2011. `http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf`.

[21] Roger Dingledine. Two relays serving same hidden service. April 2011. `http://archives.seul.org/tor/relays/Apr-2011/msg00022.html`.

[22] Python documentation: 9.6. random — Generate pseudo-random numbers. `https://docs.python.org/2/library/random.html`.

[23] Shallot, July 2012. https://github.com/katmagic/Shallot.

[24] The Tor Project, Inc. Tor Protocol Specification, 1.1. Keys and names, Feb 2015. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt#n140.

[25] The Tor Project, Inc. Tor Protocol Specification, 2. Connections, Feb 2015. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt#n157.

[26] The Tor Project, Inc. Tor directory protocol, version 3 - guard life time, Mar 2015. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt#n1617.

[27] The Tor Project, Inc. Tor directory protocol, version 3 - num entry guard, Mar 2015. https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt#n1612.

[28] The Tor Project, Inc. Tor Rendezvous Specification, 1.4. Bob's OP advertises his service descriptor(s), Feb 2015. https://gitweb.torproject.org/torspec.git/tree/rend-spec.txt#n471.

[29] Bernd Kreuss (author of TorChat). Issue 13: Implement a block list, 2008. https://code.google.com/p/torchat/issues/detail?id=13.

[30] Bernd Kreuss (author of TorChat). TorChat Download page in Github, September 2012. https://github.com/prof7bit/TorChat/downloads.

[31] Debin Linux - Package: torchat (0.9.9.553-1.1). https://packages.debian.org/jessie/torchat.

[32] Ubuntu Linux 14.04 Trusty - Package: torchat (0.9.9.553-1). http://packages.ubuntu.com/trusty/web/torchat.

[33] Ubuntu Linux 15.04 Vivid - Package: torchat (0.9.9.553-1.1). http://packages.ubuntu.com/vivid/web/torchat.

[34] Arch Linux - Package Details: torchat 0.9.9.553-2. https://aur.archlinux.org/packages/torchat/.

[35] Bernd Kreuss (author of TorChat). TorChat SVN commit r292, December 2010. https://code.google.com/p/torchat/source/detail?r=292.

[36] PyInstaller, March 2015. https://github.com/pyinstaller/pyinstaller/wiki.

[37] Bernd Kreuss. TorChat change log, September 2012. https://github.com/prof7bit/TorChat/blob/torchat_py/torchat/src/changelog.txt.

[38] Roger Dingledine. OpenSSL bug CVE-2014-0160, April 2014. https://blog.torproject.org/blog/openssl-bug-cve-2014-0160.

[39] Tor Binary Too Outdated Again #62 , October 2014. https://github.com/prof7bit/TorChat/issues/62.