A Telegram bot for printing LaTeX files

Amartyo Banerjee and S.K Venkatesan

Abstract

A proof of concept of a Telegram bot running on a Raspberry Pi is described here. The bot will accept a LaTeX file from the user, process it and send back to the user a PDF file resulting from that processing. The following are discussed:

1. The genesis of the idea of the bot.
2. The use case for the bot as it exists at present, and after additional functionality is implemented.
3. The learning process and obstacles encountered in developing it.
4. Additional functionality planned to be implemented, such as processing a multi-file LaTeX document, and printing the PDF file.
5. Steps needed to make it production ready, including robust error handling and proper input sensitization.
6. Potential for a complete rewrite to meet scalability requirements and get around file download limitations in the Telegram bot API.

1 Introduction

The authors present a proof of concept of a Telegram [1] bot [2], meant to run on the Raspberry Pi [3]. The purpose of this bot is to accept (La)TeX files submitted by a user running the Telegram client on a mobile phone or on a PC/laptop browser, and to send back a PDF file produced by compiling the (La)TeX files using pdfTeX or XeTeX or whatever (La)TeX compiler is invoked by the files.

2 Genesis of the idea

The idea for implementing this bot arose out of a brainstorming session when the various uses that could be made of the Raspberry Pi were discussed. The question that came up was if it is possible to run TeX on the Raspberry Pi. After checking online, it was found that not only could TeX be run on the Raspberry Pi, but people were in fact running it [4]. At this point we considered a potential use case for the Raspberry Pi, wherein people could type up a TeX file on their mobile phone, send it to server software running on the Pi which could compile the TeX file into PostScript or PDF and print it out. The person who had submitted the TeX file could then come and collect the typeset and printed output of that TeX file. This service could be a paid service.

Figure 1: A Raspberry Pi

3 The use case for the bot

At the moment the printing functionality is not yet implemented. Even the current functionality, however, has some use cases. A user could use (La)TeX to write a music score [5], and see what the typeset output would look like. Or they could type a chess game of a certain move in LaTeX chess notation, and get back a view of what that looks like when typeset [6].

The idea is to eventually have a system where a person can submit one or more (La)TeX files which will be processed and the resulting output will be printed out. The print can be collected later on, which would make the bot suitable for use in a print shop. So a user could submit a (La)TeX file to the bot, it would be processed into a Postscript or PDF file and this file could then be printed out. The user could then pay to collect the print.

4 The learning process and obstacles encountered

Initially the idea was for the TeX files to be submitted via SMS. However, we decided against this on the basis of our recollection of the difficulties in working with SMS messages in computer programs, especially as there are also restrictions in India on how many SMS messages can be sent by a particular number and to a particular number, as also restrictions on what sort of attachments can be sent by SMS, and on their size. Last but not least each SMS costs money.

For these reasons we decided to use messaging software that does not depend on SMS messages to send messages and attachments. In the authors' circle of friends and acquaintances, the most often used messaging applications on their mobile phones are WhatsApp [7] and Viber [8], with more WhatsApp users than Viber, and those contacts in the first author's phonebook who use both are far more likely to check WhatsApp than Viber.

This seemed to make WhatsApp the software of choice for transmitting TEX files to the proposed server software. The case seemed to be strengthened by the fact that in past searches for ideas of things that could be done with a Raspberry Pi, the first author had seen a write-up on using WhatsApp on the Raspberry Pi to send notifications to the owner of the Raspberry Pi [9] whenever events occurred which the owner was interested in knowing about.

To the best of the first author's recollection however, in the comments made on the web page [9], it had been pointed out that the use of WhatsApp in this manner involved the use of reverse engineered knowledge of the WhatsApp protocol, which is proprietary. The protocol could be changed anytime by WhatsApp, which would break any software relying on the reverse engineered understanding of the protocol. Writing such software and using it also had the potential of violating WhatsApp's terms of service, which could result in termination of the WhatsApp account of the person whose mobile phone number had been used to register the unofficial software that used the reverse engineered WhatsApp protocol.

In these same comments, it was pointed out that in contrast to WhatsApp, the Telegram messaging application had an officially documented protocol [10], permitted the existence of open source clients [11], and provided not just one but two APIs [12] which could be used to interact with Telegram and its servers, and which could be used to write software using Telegram as the messaging part of that software.

As it turns out, the first author's recollection turns out to be faulty in this case. They have been unable to find any comments stating anything listed in the above two paragraphs. Perhaps such comments had indeed been seen on that website but in that case they have subsequently been deleted. Checking in the Wayback Machine [13] for past versions of the url in [9] does show certain comments which have been deleted from the live site, but not the comments the first author seems to recall. Alternatively, the first author saw such comments elsewhere but now cannot recall where that could have been, nor have they been able to discover in their recent web searches any other website containing such comments. It is possible that in the first author's memory they have conflated their reasoning with a url where they thought they had seen someone else make those points.

That being said, the substance of the first author's objections remains valid. The url mentioned in [9] refers to the use of a python library named Yowsup [14]. This library uses a reverse engineered API for WhatsApp, named Chat API [15]. On one of the Chat API web pages [16], they mention that they have received a cease and desist letter from WhatsApp's lawyers, a copy of which they have made publicly available at the url listed in [17]. Although the author of Chat API states in [16] that they will maintain the repository mentioned whose url is [15], the letter shown in [17] does confirm that Chat API is a reverse engineering of WhatsApp's API, and that WhatsApp considers the development and use of Chat API to be a violation of its terms of service.

It is also the case that the author of the tutorial in [9] mentions partway through [18] the risks of getting one's mobile phone number banned through repeated attempts to register the same number, and recommends using Telegram.

Remembering these objections, however hazily, when the time came to choose how a potential user could send TEX files to the proposed server software running on the Raspberry Pi, it was decided to use Telegram. The authors discussed the reasoning and concerns of the first author. Given that SMS is intrinsic to every mobile phone and WhatsApp is already widely installed the second author might have insisted on using either of these. Instead the second author agreed with the first author, reasoning that the potential users they had in mind could be persuaded to install Telegram on their mobile phones, and in any case at some future date a mobile phone application could be written integrating a text editor with the ability to send TEX files to the server software. In that case potential users would have to be persuaded to install that application on their phone anyway.

As mentioned above, Telegram provides a choice of two APIs to interact with it. One is an API to write a full-fledged Telegram client [19]. Our program, if written using the Telegram client API, would thus be a client from the point of view of the Telegram servers while also being a server. In other words, it would be a client from the point of view of the Telegram servers, but a server from the point of view of our users. The service provided by it would be the fact that as mentioned above, if sent a (LA)TEX file it would return a typeset PDF and/or print said PDF.

The second API [20] allows one to write a bot for Telegram. The exact definition of what a Telegram bot is and can do can be found on the Telegram website [2, 20]. That being said, the closest analogy that comes to mind is the variety of bots [21] written for IRC [22]. In some sense it seems to the authors that Telegram bots are a repackaging of an old idea for a generation of Internet users whose primary interaction with the Internet is via smartphones, who might have never come to know about IRC and IRC bots.

On reviewing the two APIs, it seemed to the first author that writing a bot might be an easier thing to get started with, especially if one wanted to get a proof of concept implementation up and running quickly. It helped that a write-up on how to implement a Telegram bot on the Raspberry Pi [23] was easily found. Before going any further it should be mentioned that a third possible way exists, which is to use a command line desktop/laptop Telegram client [11], which would run on the Raspberry Pi, and could be configured to perform certain tasks on certain events, such as the receipt of a file from a user, etc. This approach has both advantages and disadvantages. The disadvantages led to the bot approach being chosen, but the advantages might lead to the software being implemented using the command line Telegram client after all in a future iteration.

Just as in the case of running a Telegram bot on the Raspberry Pi, it was easy to find web pages giving instructions on using the Telegram command line client on the Raspberry Pi [24], as a means of controlling the Raspberry Pi remotely [25] and receiving notifications from the Raspberry Pi when certain events happened. Although we worked through the tutorials listed on those web pages, it was not immediately obvious how to extend the examples given to achieve what we wanted to achieve.

In contrast, the web pages providing instructions on implementing a Telegram bot on the Raspberry Pi [23] lead to a GitHub page containing the software used to implement the bot [26], which in turn had slightly more complex examples [27, 28] that could be extended to achieve what we wanted. The fact that the bot examples were written in Python, which the first author is more familiar with, as compared to Lua, which the Telegram command line client examples were written in, was also a factor that led the first author to choose the bot approach. Yet another factor in choosing the bot approach was the fact that using the Telegram client required the first author to use their mobile phone number to register with the Telegram server [24], as compared to a bot, which does not require a mobile phone number [20, 23].

Nevertheless, there is one advantage of using the Telegram command line client which may lead to it being used in a future iteration of the software. The advantage is that as per the existing Telegram bot API, the maximum size of a file that may be downloaded by a bot from the Telegram servers is 20MB [29]. This means of course, that the maximum size of a file that can be sent by a user of our software is 20MB. For the command line client, as also official Telegram clients released for various platforms, the size limitation is much higher, perhaps even as much as an order of magnitude.

Now, 20MB seems like more than enough for (LA)TEX files, which are after all like source code. However, one can never predict how large a (LA)TEX file a user might wish to compile and print. Furthermore, this 20MB limit can be reached much more quickly if the user chooses to make references to figures and other graphics files in their (LA)TEX files, to be included in the final typeset PostScript/PDF file.

During discussions, the second author stated that they did not anticipate many of the users they had in mind needing to exceed that 20MB limit, so for now the bot approach is the only implementation that exists. If the software is found generally useful, a new implementation using the command line client approach will be done in future.

The implementation of the code took a few days. Much of this time was spent working through the tutorial for implementing a Telegram bot on the Raspberry Pi [23], trying to understand and then extend the more complex examples given on the website of the software used to implement the bot API [27, 28, 30], known as telepot [26], trying to understand the Telegram bot API [31] per se, and simply brushing up the first author's Python and general programming knowledge.

Further time was taken up when the function provided by telepot to download a file [32] was found not to work. In this case that meant the code would hang at that point, until one was forced to kill the bot. Initially a workaround was coded whereby `wget` [33, 34] was used to directly download the file from the Telegram servers. After digging into the implementation of the file download function, and doing some research online [35], using comments in the telepot source code itself [36], we were able to patch the telepot source files to make the download file function work correctly. The patch itself is trivial although the effort to figure out what to do was not. It will be submitted to the author of telepot.

There was also the time involved in searching for LATEX server implementations, i.e. some software that listens on the network, receives LATEX files from clients and does something with it, either compile it and send back the typeset output or print it or something else. We found a few interesting links but nothing that fit our purpose [37–41]. On the other hand, those web searches did provide pointers to what was eventually implemented [42, 43].

5 User experience and implementation

As currently implemented, the user will have the following experience when interacting with the bot.

First, the prerequisites. The user will have to download and install the official Telegram client for their smartphone [44–46]. There is a registration process which requires providing a mobile phone number, which is used by the Telegram server for verification and to complete the installation of the client program. This is similar to the process used to install other mobile phone centric messaging applications like WhatsApp and Viber. Any user who has successfully installed and used these other applications to communicate with others should have no problem completing this process on Telegram.

After this, they have to search for the bot, named AmartyoFirstBot. On selecting this bot from the search results, initially a button labelled 'Start' will appear, which has to be pressed. According to the bot API documentation [47], this is one of the commands to which the bot must give a response, but at present it does nothing. At this point a chat session will be opened with the bot. The interface at this point is similar to opening a chat with a user on the other messaging applications mentioned earlier.

Now the user has to select an attachment to send, by pressing the button on the phone touchscreen that resembles a paper clip. This brings up a series of icons, which are meant to select photos, videos, music, voice clips and documents. The icon to send documents must be selected, and then a LaTeX file selected using the file manager interface. Once selected the user is taken back to the chat session, and an icon appears the pressing of which sends the LaTeX file to the bot.

It must be emphasized at this point that the process of installing Telegram, searching for a contact or another Telegram user or a bot, is identical for every Telegram user whether or not they ever interact with our bot. The same applies to starting a chat session, either with a user or a bot, and for selecting an attachment to send and sending it.

What is unique in our case is that when a LaTeX file is sent, the user will ultimately see that a bot has sent a PDF file as a message, and will have an option to download it. On downloading it, they can tap on the icon and the PDF file will display in whichever application is configured to display PDF files.

On the server side, the bot, on receiving a chat message, checks if it has been sent a document, and if that document is a LaTeX file. If so, it downloads that file and saves it in a temporary directory. It then changes to that directory and invokes the `latexmk` [48] command on the received LaTeX file, using Python's facility for calling external programs. In the invocation it passes certain parameters to `latexmk`, along with the name of the LaTeX file. One

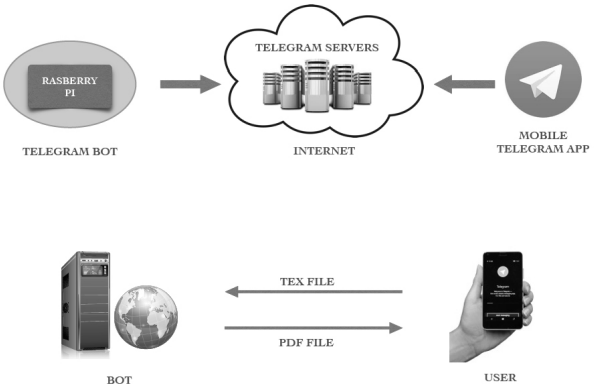

Figure 2: The LaTeX bot

of these specifies that `latexmk` should try to create a PDF file, which by default `latexmk` does by invoking the `pdflatex` [49] command. The bot checks the return code of `latexmk`, and if the return code indicates that `latexmk` succeeded, it proceeds to send the PDF file created by `latexmk` to the user who had sent the LaTeX file. At this point the bot changes back to the directory it was in before it received the LaTeX file. It continues running, waiting for messages.

6 Future functionality

As mentioned above, the bot is strictly a proof of concept at this point. It lacks functionality and also robust error handling, not to mention any effort at sanitizing what it receives from the user. In terms of functionality, the most obvious thing missing is the ability to handle anything more than a single TeX file. Thus, a user who has written a thesis or report in LaTeX, which typically include a single master TeX file with reference to multiple other LaTeX files, each containing either the text of a chapter, or maybe references to other LaTeX files, as well as figures and other pictures, will not be able to use the bot to compile and print that report or thesis.

The authors have an idea for implementing this functionality. It involves allowing the user to send the bot a zip file containing all the LaTeX and other files needed to compile and produce a report, such as figures. It will be the user's responsibility to make sure the folder structure inside the zip file corresponds to the declarations in the main LaTeX file, such that invoking `latexmk` on the main file will successfully find all the other LaTeX files and any (Encapsulated) PostScript or other graphics files needed, as well as any non-standard fonts or other files. If the user wishes to use fonts beyond those that every TeX installation is expected to have by default, it will be their responsibility to include those

font files inside the zip file at the correct path such that `latexmk` can find them.

The bot will expect the name of the zip file, the part before the `.zip` extension, to be the same as the name of LaTeX file inside it. This LaTeX file containing the same name as the zip file will be expected to be the main LaTeX file, on which `latexmk` will be invoked. On successful compilation, the resulting PDF file will be sent back to the user and/or printed.

Other ideas for functionality include options for letting the user indicate if they wish to simply receive the PDF file or to actually print it. In the latter case, they will initially receive the PDF file to verify that the output is as they expect, and then an option to approve the final print. This will be important in case the bot is used in a business, such as a print shop. Requiring the user to approve the PDF before printing should help in avoiding disputes where the user claims the printed output does not look like they wanted it to and refuses to pay for the print.

At this point it is not clear how this "approve before printing" functionality will be implemented. The Telegram bot API already provides a variety of information such as the user id, the chat session id, the message id, etc. This will have to be kept track of in some sort of database, using which the bot will be able to know whom it received a particular TeX or zip file from, that the compilation succeeded and the PDF file was successfully sent back to that user, and that the user has approved it and agreed to print it. Some sort of record will have to kept of the fact that a user has approved a printout. On the user end, perhaps a combination of custom keyboards, which is functionality provided by the Telegram bot API, can be used to provide a user interface for this approval functionality.

Other functionality that needs to be implemented is the `/start` command, which every bot is expected to implement, along with a few others [47].

7 Becoming production ready

As far as error handling is concerned, at present if `latexmk` returns a non-zero error code, the bot simply exits with an error code of `1`. This is obviously not suitable for production purposes. If `latexmk` fails to compile the TeX document, the bot should send relevant error messages back to the user, to enable the user to correct whatever errors caused `latexmk` to fail, whether in the syntax of a single LaTeX file, or in the paths where other LaTeX files or graphics/font files are supposed to exist.

It might also be the case that `latexmk` fails because the user intended to produce PostScript as the final output, not PDF, and that pdfLaTeX has failed because the LaTeX files refer to PostScript files for use as figures, not Encapsulated PostScript. In that case, since PostScript is perfectly suitable for printing, the bot should attempt to invoke `latexmk` with the option to produce a PostScript file as the final output, not PDF. Software for viewing PDF files is widespread nowadays, but this is not the case for PostScript files. For the purpose of sending something back to the user, the PostScript file produced by `latexmk` should be converted using `ps2pdf` [50], and this PDF file can then be sent by the bot. On the other hand, the PostScript file produced by `latexmk` should be the one sent to the printer by the bot.

Similarly, the user may have intended for their files to be compiled by XeLaTeX [51, 52] rather than pdfLaTeX, in which case the bot should invoke `latexmk` appropriately.

Perhaps one approach is to invoke `latexmk` by default with the `-pdf` option, then try with the `-xelatex` option, and if that fails try with the `-ps` option. If all these fail, the bot should give up and send an appropriate error message back to the user.

A more tricky case arises in cases where `latexmk` returns 0, but the resulting PDF still does not contain what the user intended. This can happen for example when trying to typeset music using LaTeX and the `abc` [53] notation. We noticed that if there was a syntax error in the `abc` notation within the LaTeX file [54, 55], a PDF file might get successfully created but with the actual typeset music missing. This is because in this case `latexmk`, or even `pdflatex`, invokes another program called `abcm2ps` [56, 57], which fails to compile the erroneous `abc` syntax. The `abcm2ps` program does print error messages indicating that it has failed. However, it either does not exit with appropriate error codes, or those are ignored by `pdflatex` and/or `latexmk`, most likely `pdflatex`. As a result `latexmk` returns 0 to indicate success when in fact this is not the case.

Perhaps one option is to check if the LaTeX file is using the relevant LaTeX package for `abc` notation, and in that case check the output of `latexmk` for the known error messages by `abcm2ps` indicating that it has failed. In which case the misleading zero exit code of `latexmk` should be ignored, and the bot should send back the relevant error messages of `abcm2ps`. However, this will make the bot code more complicated, so perhaps a better approach is to make changes to either `abcm2ps` or `pdflatex`, which ever it needs to be, such that if `abcm2ps` fails then `pdflatex` should exit with a non-zero value.

What is to be done if `latexmk` received a signal [58] causing it to exit uncleanly is not clear to the authors at the moment. It seems to the first author

at present that the options are for the bot to treat it as a transient error and try invoking `latexmk` again, or assume that something is seriously wrong with the system it is running on, try to send an appropriate error message to the user, and exit as cleanly as it can. More thought and discussion is required before any decision can be taken.

Last but not least, to be production ready, the bot must implement sanitization of inputs submitted by the user. This is a necessity for any internet facing server software in this era of SQL injection attacks [59], cross site scripting attacks [60], privilege escalation [61] and arbitrary remote code execution vulnerabilities [62], which are often made possible by not sanitizing or improperly sanitizing user inputs [63, 64], especially when provided by some unknown and untrusted user over the Internet. Measures to be taken that come to mind immediately are making sure the names of files submitted by the user are sane, and that certain metadata needed by the bot are present, even in cases where the bot API says such metadata is optional [65]. Other measures include making sure that the contents of files submitted by the user match their claimed mime type, and that they are not in fact viruses and/or some other malware. This is a subject in which the first author does not have much expertise, and more research is needed to ensure that the bot implements proper input sanitization.

8 Potential for a complete rewrite

In the long run the bot may need to be re-written in Python 3, simply to take advantage of the fact that telepot provides an API to write asynchronous code, which might well be a necessity for scalability reasons at some point in the future, but only for Python 3 [66]. This will involve learning about the differences between Python 2 and Python 3, and about how to program asynchronously in Python 3, and then how to use telepot in an asynchronous manner. This will take quite a lot of time, so we only expect to do it if and when the bot becomes popular enough that scalability issues matter. At any rate, it will be done after implementing necessary functionality and fixing the error handling and sanitizing user inputs as listed above.

9 Epilogue

At the time of completion of the first draft of this paper, the bot had been written and was running on the first author's laptop. The Raspberry Pi on which the bot was meant to run was delivered to the first author on 10th June, 2016. Subsequently the Raspberry Pi was powered up and made to run and the prerequisites for getting the bot running on the Raspberry Pi were installed and configured. As of the current date the bot is installed on the Raspberry Pi. It currently needs to be started manually, once the Raspberry Pi is powered up.

Making sure the bot starts running automatically upon powering up the Raspberry Pi is one of the improvements to be done in future, along with all the other improvements outlined above, including printing functionality.

References

[1] telegram.org
[2] core.telegram.org/bots
[3] www.raspberrypi.org/help/what-is-a-raspberry-pi
[4] www.raspberrypi.org/forums/viewtopic.php?f=63&t=8279
[5] martin-thoma.com/how-to-write-music-with-latex
[6] www.highschoolmathandchess.com/2011/10/06/creating-chess-diagrams
[7] www.whatsapp.com
[8] www.viber.com/en
[9] www.instructables.com/id/WhatsApp-on-Raspberry-Pi/?ALLSTEPS
[10] core.telegram.org/mtproto
[11] github.com/vysheng/tg
[12] core.telegram.org/api
[13] archive.org/web
[14] github.com/tgalal/yowsup
[15] github.com/mgp25/Chat-API
[16] github.com/mgp25/Chat-API/wiki/WhatsApp-incoming-updates#11-july-2015
[17] www.docdroid.net/gWpFsXz/whatsapps-cease-and-desist-and-demand-against-chat-api.pdf.html
[18] www.instructables.com/id/WhatsApp-on-Raspberry-Pi/step2/Registration
[19] core.telegram.org/api#telegram-api
[20] core.telegram.org/api#bot-api
[21] en.wikipedia.org/wiki/IRC_bot
[22] en.wikipedia.org/wiki/Internet_Relay_Chat
[23] www.instructables.com/id/Set-up-Telegram-Bot-on-Raspberry-Pi/?ALLSTEPS
[24] www.instructables.com/id/Telegram-on-Raspberry-Pi/?ALLSTEPS
[25] www.instructables.com/id/Raspberry-remote-control-with-Telegram/?ALLSTEPS

[26] github.com/nickoala/telepot
[27] github.com/nickoala/telepot/blob/master/examples/simple/skeleton.py
[28] github.com/nickoala/telepot/blob/master/examples/simple/skeleton_route.py
[29] core.telegram.org/bots/api#getfile
[30] github.com/nickoala/telepot/blob/master/doc/reference.rst
[31] core.telegram.org/bots/api
[32] github.com/nickoala/telepot/blob/master/telepot/__init__.py#L460
[33] en.wikipedia.org/wiki/Wget
[34] www.gnu.org/software/wget
[35] stackoverflow.com/questions/17285464/whats-the-best-way-to-download-file-using-urllib3
[36] github.com/nickoala/telepot/blob/master/telepot/__init__.py#L473
[37] alex.nederlof.com/blog/2013/02/22/latex-build-server
[38] web.archive.org/web/20120120210031/http://bugsquash.blogspot.com/2010/07/compiling-latex-without-local-latex.html
[39] scribtex.wordpress.com/2010/01/17/the-common-latex-service-interface
[40] github.com/jpallen/clsi
[41] launchpad.net/rubber
[42] superuser.com/questions/173914/dropbox-latex-automated-pdf-compile
[43] latex-community.org/forum/viewtopic.php?f=28&t=9512
[44] telegram.org/dl/android
[45] telegram.org/dl/ios
[46] telegram.org/dl/wp
[47] core.telegram.org/bots#global-commands
[48] users.phys.psu.edu/~collins/software/latexmk
[49] tug.org/applications/pdftex
[50] www.ghostscript.com/doc/current/Ps2pdf.htm
[51] xetex.sourceforge.net
[52] tug.org/xetex
[53] abcnotation.com
[54] martin-thoma.com/how-to-write-music-with-latex/#abc
[55] martin-thoma.com/how-to-write-music-with-latex/#example
[56] moinejf.free.fr
[57] abcplus.sourceforge.net/#abcm2ps
[58] en.wikipedia.org/wiki/Unix_signal
[59] en.wikipedia.org/wiki/SQL_injection
[60] en.wikipedia.org/wiki/Cross-site_scripting
[61] en.wikipedia.org/wiki/Privilege_escalation
[62] en.wikipedia.org/wiki/Arbitrary_code_execution
[63] en.wikipedia.org/wiki/Improper_input_validation
[64] xkcd.com/327
[65] core.telegram.org/bots/api#document
[66] github.com/nickoala/telepot#async

◇ Amartyo Banerjee and S.K Venkatesan
TNQ Books and Journals
Chennai, India
http://tnqsoftware.co.in