System & Network Engineering - University of Amsterdam
Research Project 1

Breaking CAPTCHAs on the Dark Web

11 February, 2018

Authors
K. Csuka
kevin.csuka@os3.nl

D. Gaastra
dirk.gaastra@os3.nl

Supervisor
Y. de Bruijn
yonne.debruijn@fox-it.com

Abstract

On the Dark Web, several websites inhibit automated scraping attempts by employing CAPTCHAs. Scraping important content from a website is possible if these CAPTCHAs are solved by a web scraper. For this purpose, a Machine Learning tool is used, TensorFlow and an Optical Character Recognition tool, Tesseract to solve simple CAPTCHAs. Two sets of CATPCHAs, which are also used on some Dark Web websites, were generated for testing purposes. Tesseract achieved a success rate of 27.6% and 13.7% for set 1 and 2, respectively. A total of three models were created for TensorFlow. One model per set of CAPTCHAs and one model with the two sets mixed together. TensorFlow achieved a success rate of 94.6%, 99.7%, and 70.1% for the first, second, and mixed set, respectively. The initial investment to train TensorFlow can take up to two days to train for a single type of CAPTCHA, depending on implementation efficiency and hardware. The CAPTCHA images, including the answers, are also a requirement for training TensorFlow. Whereas Tesseract can be used on-demand without need for prior training.

1 Introduction

The dark web contains, among other things, information about illegal activity, which is of interest to various organizations from an intelligence perspective [1]. The intelligence can be used to monitor activity related to specific high profile organizations, or specific threat actors. Since there are a lot of different types of websites, with sometimes unique subscriber requirements it is hard to scrape these websites. In some cases, an existing member has to vouch for a new member, users have to post at least once a month a message on the website, or one has to pay in Bitcoin to get access, etc.. The goal of this research is to come up with a theoretical framework and provide a Proof of Concept for scraping potentially interesting dark web websites that inhibit scraping through the use of CAPTCHAs.

The Turing Test scenario is based on an interrogator chatting remotely with a machine and a human. The interrogator asks several questions, to both human and machine, and inspects their answers. If the interrogator can accurately differentiate between human and machine by virtue of the humans answers being more accurate, then the investigated system is not intelligent, and vice versa [2].

CAPTCHA is an abbreviation which stands for 'Completely Automated Public Turing test to tell Computers and Humans Apart'. As the acronym suggests, it is a test to determine whether the user is human or not.

The main purpose of CAPTCHAs is to distinguish between humans and machines in security applications, such as preventing automatic comments and registration on different websites, protecting email addresses from web scraping or protecting online polls from being skewed by certain views [3].
There are three common approaches to defeat CAPTCHAs:

1. Using a service which solves CAPTCHAs through human labor
2. Exploiting bugs in the implementation that allow the attacker to bypass the CAPTCHA
3. Character recognition software to solve the CAPTCHA

In this research we focus on the third item, using software to solve a CAPTCHA.

1.1 Research questions

Based on this information, the research question is defined as:

- **How would a scraper be able to circumvent CAPTCHAs that prevent it from properly scraping dark web websites?**

In order to answer the main research question, the following sub-questions are defined:

- What will be the potential impact of breaking CAPTCHAs on the dark web?
- How can standard Optical Character Recognition (OCR) software aid a scraper in circumventing CAPTCHAs?
- How can Machine Learning and Neural Networks aid a scraper in circumventing CAPTCHAs?

2 Related work

While CAPTCHAs have been developed based on pure text, images, audio and video, text CAPTCHAs are almost exclusively used in real applications, such as dark web websites. In a text CAPTCHA, characters are deliberately distorted and connected to prevent recognition. Security of an existing text CAPTCHA is enhanced systematically by adding noise and distortion, and arranging characters more tightly [4].

Lawrence et al. created their own dark web scraping tool, named D-miner [5]. They describe it is important to maintain anonymity while browsing marketplaces as it is possible for operators to detect browser settings unique to individuals in order to fingerprint users. The Tor Browser is a standard web browser maintained by the Tor Foundation for the purpose of browsing onion websites available over the Tor network. The Tor Browser attempts to mitigate fingerprinting by setting a static resolution of the browser and adjusting multiple settings upon starting. Similarly, D-miner uses Selenium to programmatically adjust its fingerprint to appear as a normal Firefox browser. D-miner then authenticates to the dark web site and hands the session to the Requests library thus utilizing the headers and authentication data from Selenium. Therefore, even after scraping is handed off from Selenium to Requests, D-miner has the same fingerprint as Firefox.

D-miner has no functionality included to bypass CAPTCHAs. It solves this by human labor. According to the authors: "Programmatic approaches to captcha-solving are out of the scope of this research and an open research problem as many current implementations are largely ineffective and slow, increasing the chance of detection, the load on the DNM, and the length of time needed to scrape." [5] DNM refers to DarkNet Marketplace.

Ryan Mitchell demonstrates that Python is a fantastic language for image processing by using the library Tesseract. But in order to process images correctly, they have to be cleaned first. Cleaning can be done using the library Pillow [6]. The method described to break the CAPTCHA is Optical Character Recognition (OCR). OCR is the recognition of printed or written characters by a computer. It enables one to convert different types of documents, such as scanned paper documents, PDF files or images captured by a digital camera into editable and searchable data. Tesseract is an open-source OCR engine for various operating systems.

As Osadchy et al. researched, CAPTCHAs are most likely to be broken by machine learning. Recent advances in Deep Learning (DL) allow for solving complex problems that used to be considered very hard. While this progress has advanced many fields, it is considered to be bad news for CAPTCHAs, the security of which rests on the hardness of some learning problems. The paper introduced a new method even AI cannot solve, named DeepCAPTCHA, a new and secure CAPTCHA scheme based on adversarial examples, an inherit limitation of the current DL networks. They implemented a proof of concept system, and its analysis shows that the scheme offers high security and good usability compared with the best previously existing CAPTCHA [7].

Some naïve pattern recognition algorithms are introduced by J. Yan and A. Ahmad [8]. They focused on four visual schemes, used by captchaservice.org. The first is called word_image, which is a distorted image of a six-letter word. Secondly, they attacked random_letter_images, which consists of a distorted image of six random letters. The user_string_image is where the challenge is a distorted image of a user-supplied string with at most 15 characters. Lastly, they managed to attack number_puzzle_text_images, which is a multi-model scheme that includes distorted images along with an instruction for the user to solve a puzzle. All of these schemes were successfully broken by exploiting design flaws in them. Their success rate neared 100%.

To break CAPTCHAs, one can use Deep Learning. Arun Patala shows the ease of the tool Torch, which is a Deep Learning framework [9]. A success rate of 92% is achieved breaking simple CAPTCHAs.

3 Methods

The first part of our research consists of categorizing dark web websites. This is done to evaluate the impact of this research on privacy and security of the dark web. A total of 634 onion links are manually analyzed for this purpose.

The second part of our research aims to break CAPTCHAs to allow for scraping of the blocked parts of a website. Two approaches are analyzed and discussed. The first approach is using OCR software to recognize the characters within a CAPTCHA. The second approach makes use of Machine Learning to train a Neural Network on CAPTCHAs.

3.1 Categorizing dark web websites

The websites are categorized by their state, alive or down. A website is marked as down if it was either unable to serve a web page due to a connection error or if some form of notice is displayed that the website has moved to a different location. A website is considered alive if there is content displayed that is appropriate for the type of website. Note that no differentiation is made between the different types of websites.

Following this preliminary analysis, duplicates are identified and not taking into consideration, since a significant number of websites are reachable through multiple .onion addresses.

The blockades found on an alive, non-duplicate website are categorized as follows:

- CAPTCHA
- Closed Registration

  No CAPTCHA required

  CAPTCHA required
- Open Registration

  No CAPTCHA required

  CAPTCHA required

  Accepting the Terms and Conditions
- Accepting the Terms and Conditions
- Proof of Work
- Forbidden, HTTP error code 403
- Unauthorized, HTTP error code 401

Websites that were completely open to scraping are not categorized. This research focuses on websites which use a CAPTCHA as a blockade. For this reason, the remaining data is available in the raw data set, but not present in this paper.

3.2 Breaking CAPTCHAs

The first tool that is used to solve CAPTCHAs is Tesseract [10], which uses OCR to identify characters. It is considered one of the most accurate OCR engines currently available, with the precision depending on the clearness of the image.

The second approach is using TensorFlow [11], which creates a Neural Network model and trains it by Machine Learning. More specifically, we use large parts of P. Li's implementation of TensorFlow for CAPTCHA recognizing [12]. Once a model is trained, it can be employed to recognize patterns in new CAPTCHAs it is given. The specifications for the machine that is used to train the model are given in Appendix A.

The Neural Network is trained with two sets of CAPTCHAs [13] [14] which are used on the dark web [15] [16]. First, two models are trained, one for each set of CAPTCHAs. The training size for this is 100,000 images. Lastly, a multi-purposed model is trained to be able to recognize both types of CAPTCHA. The training size for this is includes 100,000 images of set 1 and 100,000 images of set 2 for a total of 200,000 images. The test set is still composed of 1,000 images, 500 random images from each set.

Three examples of each set of CAPTCHA are shown in Figure 1 and Figure 2.

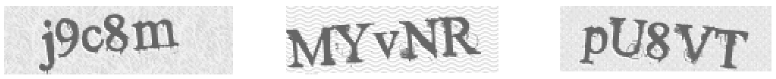


Figure 1: Three CAPTCHA examples of set 1 [13]

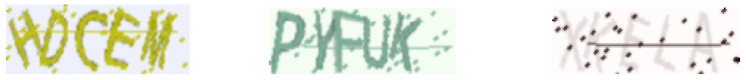


Figure 2: Three CAPTCHA examples of set 2 [14]

Each CAPTCHA contains five characters. Where every character can be either an uppercase letter, a lowercase letter, or a number. To increase readability by humans, the letters 'O', 'o', and the number '0' is omitted from the possible characters.

Both TensorFlow and Tesseract will solve the same test set, which consists of 1,000 unique CAPTCHAs. Both tools will be compared to each other in terms of success rate and accuracy. The success rate provides a measure of whether, and how often, a hypothetical implementation of the given approach would have passed the CAPTCHA. Each CAPTCHA in the test set will get a binary evaluation which is either a pass or a fail. This means that if at least one character is incorrect, the whole CAPTCHA is marked as a fail. Determining the accuracy of each tool is a more nuanced method. For each CAPTCHA in the test set, a Levenshtein algorithm [17] is used to measure the accuracy of each individual character.

The tool used to scrape the dark web is Scrapy [18]. Scrapy is the web crawler to scrape websites. It is an open source and collaborative framework for extracting data from the web and written in Python.

The workflow to solve a CAPTCHA with TensorFlow, via Scrapy, is shown in Figure 3.

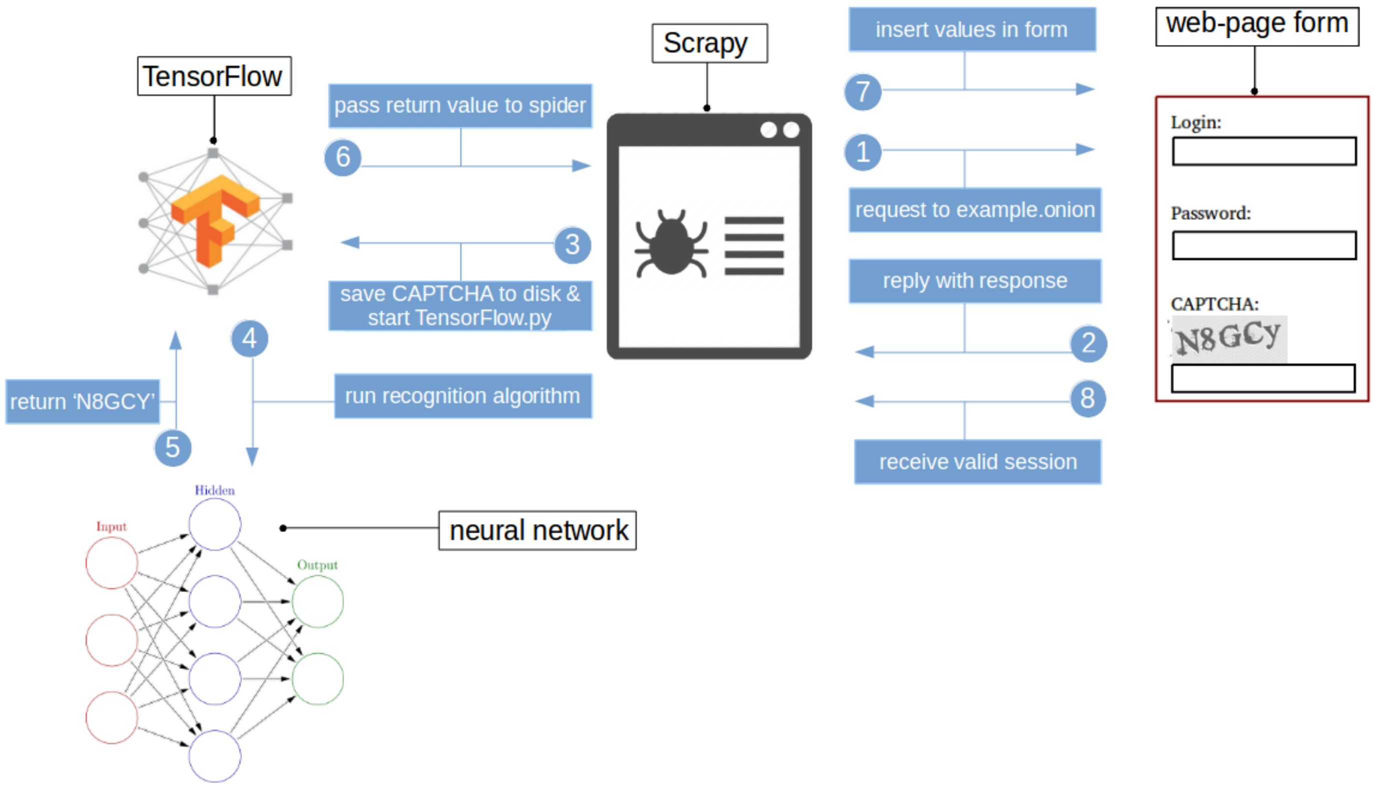


Figure 3: Workflow of solving a CAPTCHA with TensorFlow via Scrapy

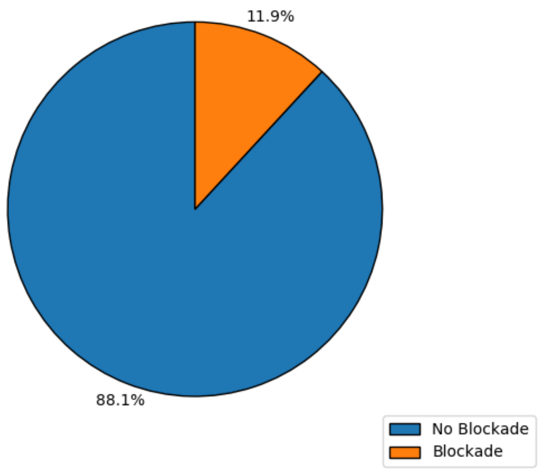
4 Results

4.1 Categorizing dark web websites

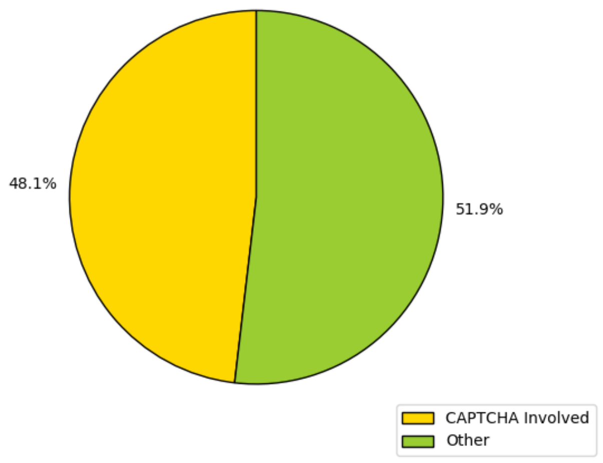
We analyzed a data set of 634 dark web websites, of which 465 were up. The data shown in Figures 4a and 4b is derived.

Figure 4a shows that almost 12% of dark web websites inhibit scraping in some way, this amounts to 55 websites in the data set. Of those websites, 51.9%, which are 26 websites, use CAPTCHAs, as shown in Figure 4b.

Modern CAPTCHAs, such as reCAPTCHA were not identified. The reason is that reCAPTCHA sends traffic to Google servers [19]. Simple CAPTCHAs are generated on the server itself and does, therefore, not have to communicate with external servers or parties.



(a) Percentage of scraping blockades encountered (n = 465)



(b) Percentage of scraping blockades using CAPTCHAs (n = 55)

Figure 4: Success rates and accuracy

4.2 Breaking CAPTCHAs

The source code of this project is available at the GitLab of OS3 [20]. The repository contains:

- A Scrapy spider, to login at a website and solving a CAPTCHA with TensorFlow
- A Scrapy spider, to solve a CAPTCHA with Tesseract
- TensorFlow, including a trained Neural Network model
- The source code of the test websites
- Script to download CAPTCHA images to disk
- Script to measure success rate

TensorFlow is able to solve the CAPTCHA and pass the return value to Scrapy. With this method, Scrapy is able to circumvent the blockade and thus log in.

It took TensorFlow 68 hours to complete training a data set consisting of 100,000 CAPTCHA images. The specifications hardware of the server used are provided in Appendix A. During training, TensorFlow reached 50,780 steps. Meaning, there was a total of 50,780 iterations made. Every Step had a batch size of 128 images. Every 10 steps, the TensorFlow loss function is run. This function returns a value that states the amount of errors that were made in the Neural Network. The loss over the steps taken is plotted in Figure 5. Note that for a proper scale, the loss values above 5 were omitted. Therefore, the step count starts at 30 instead of 0.

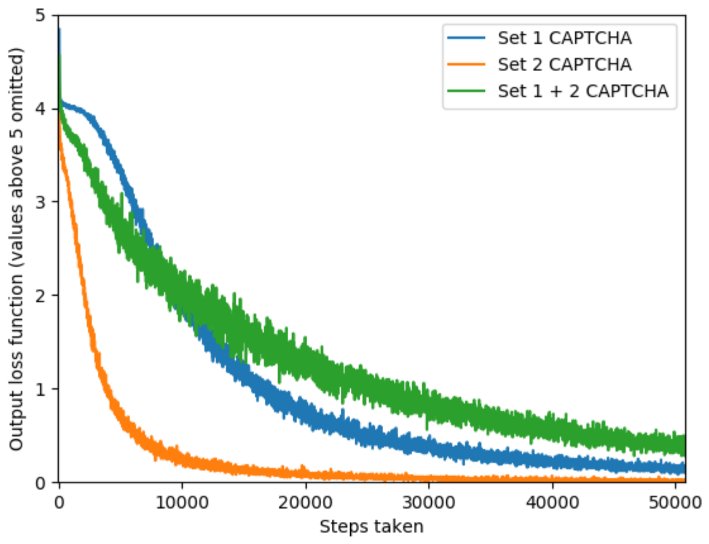


Figure 5: Loss over steps

The measured success rate is shown in Figure 6. Tesseract successfully solved 276 CAPTCHAs on set 1 and 137 on set 2, performing considerably worse on that type of CAPTCHA. TensorFlow successfully solved 946 CAPTCHAs on set 1 and 997 on set 2. On a mixed set, TensorFlow scored worse than with the individual sets with the amount of CAPTCHAs solved being 701.

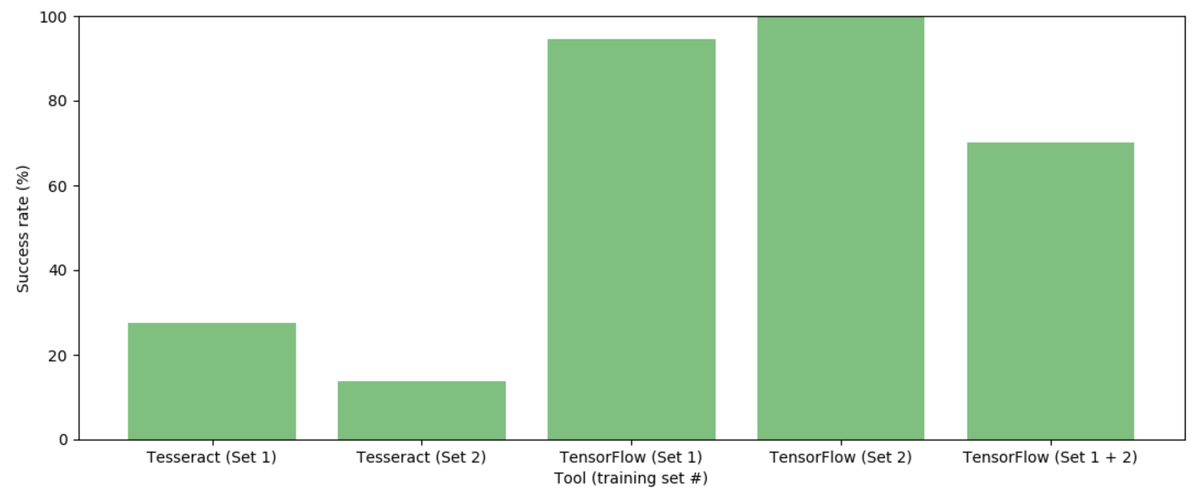


Figure 6: Correctly solved CAPTCHAs, higher is better

The total count of Levenshtein distance is shown in Figure 7. TensorFlow is the most accurate with a Levenshtein distance of 69 on set 1, 3 on set 2, and 457 on the mixed model. Tesseract achieved a Levenshtein distance of 2178 on set 1 and 3629 on set 2.

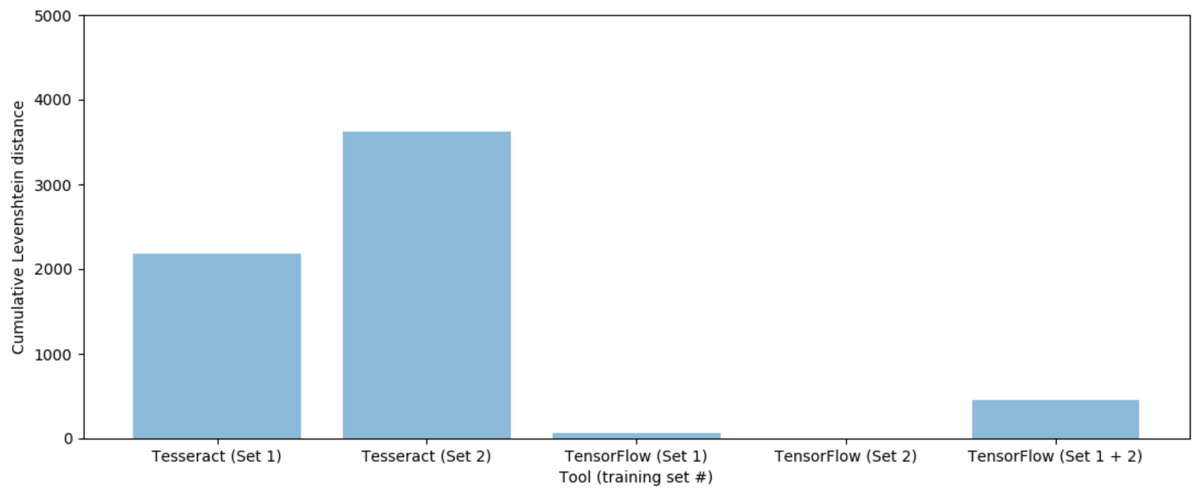


Figure 7: Total count of Levenshtein distances, lower is better

5 Discussion

Out of the alive, non-duplicate 465 onion sites investigated, 26 of these sites blocked scraping by using CAPTCHAs. This number may seem low, but it does not take away the fact that the content on these websites may be of interest.

The current implementation of TensorFlow can be trained for simple CAPTCHAs resulting in a high success rate. Models that are trained on a mixed set of different CAPTCHAs seem to be performing worse than their single set counter parts. Furthermore, training a model will take time. However, training can faster using better hardware or training on GPU instead of CPU. Furthermore, as shown in Figure 5, the training process can be halted between 10,000 and 40,000 steps, depending on the difficulty of the CAPTCHA being trained.

Tesseract shows a much lower success rate and accuracy. This also depends heavily on the CAPTCHA type it is trying to interpret. However, it should be noted that Tesseract is faster to deploy, as no model needs to be trained beforehand. In most cases, a high success rate will not be necessary since the spider will be able to make multiple requests to the server.

TensorFlow requires a data set for training a model. The model cannot be trained if a CAPTCHA does not include an answer. Therefore, having access to the source code of creating a CAPTCHA is highly recommended, since this allows the generation of a CAPTCHA with the corresponding answer.

Some CAPTCHAs on the dark web cannot be recreated by available tools since it makes use of proprietary code. Downloading a CAPTCHA and filling in the answer can be done manually, but might be undesired since it could take up a lot of time. An option to save time is to use a service which solves the CAPTCHA [21].

To give an indication, solving 1,000 images costs $1,39. TensorFlow could be trained with a set of about 50,000 images, depending on the difficulty of the CAPTCHA, resulting in a cost of $69.50 per CAPTCHA set. However, as shown in Figure 5 the actual amount for a decent success rate also depends on the CAPTCHA itself.

All in all, it should be noted that the results of this research can have big implications on web sites that do not want to be scraped. For this reason, we suggest using other, smarter, methods to aid the prevention of unwanted scraping on a web site. Such methods include, but are not limited to: behavior-based blocking and authorization of users with a permissions-based system.

6 Future work

Due to the scope, this research was unable to provide a more granular study of the websites analyzed in the first part of the research. A more granular approach could involve, but is not limited to: identifying websites of interest, a deeper study into the content of a website, and an analysis of privilege structures in place that could inhibit a human user from seeing certain content.

The implementation of TensorFlow used in the second part of our research did not apply character segmentation, since this would result in an approach that was less universal. However, future research can be conducted in this field as it could increase accuracy and success rate. A possible framework could be Tesseract 4.0, which from this version on uses Machine Learning as well, but is currently still in the Alpha development stage [22].

Improvements can be made to the Tesseract implementation. The current approach feeds images into Tesseract without applying improvements to the images. Improvements such as reducing noise, enhancing the contrast and rotating the image, so the characters are upright, could result in a higher success rate and accuracy.

The optimal training time, training data, and batch size for TensorFlow can be further investigated for different use cases.

The scraper is currently configured for one use case, it solves one type of CAPTCHA used on one website. If this technology is to be used in a production environment, the scraper should be developed for more use cases, such as preventing the scraper from being detected by a web server. This can be done, for example, by spoofing a user-agent or configuring the spider to crawl at different crawl speeds.

7 Conclusion

Our research shows that the most accurate solution for CAPTCHA solving is using a Machine Learning tool, such as TensorFlow. Even when the accuracy differs per CAPTCHA set, TensorFlow shows a high accuracy rate at each set. A success rate of almost 100% was achieved by employing Machine Learning techniques. However, if immediacy takes precedent over success rate and accuracy, then Tesseract might be a better option.

TensorFlow allows combining multiple CAPTCHAs into one model, allowing for a more versatile solution at the cost of some accuracy.

Active scraping blockades are put up by numerous websites, about half of which involve a CAPTCHA in their blocking measures. Since TensorFlow, Tesseract, and Scrapy are both written in Python. Our Proof of Concept demonstrates that it is fairly easy to combine these products to bypass CAPTCHAs.

Training a model for TensorFlow requires CAPTCHAs and the answer of the CAPTCHA. Therefore, it is advised to own the source code of creating a CAPTCHA.

8 Acknowledgements

The authors would like to thank Yonne de Bruijn from Fox-IT for his active support and supervision during this project.

References

[1] Krijn de Mik. RP 2017-2018. http://delaat.net/rp/2017-2018/index.html.

[2] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

[3] Ahmad Hassanat. Bypassing CAPTCHA By Machine A Proof For Passing The Turing Test. *arXiv preprint arXiv:1409.0925*, 2014.

[4] Sushma Yalamanchili and M Kameswara Rao. A framework for devanagari script-based captcha. *arXiv preprint arXiv:1109.0132*, 2011.

[5] Jeff Yan and Ahmad Salah El Ahmad. D-miner: A framework for mining, searching, visualizing, and alerting on darknet events. In *Heather Lawrence, Andrew Hughes, Robert Tonic, Cliff Zou*, pages 1 – 9. IEEE, 2017.

[6] Ryan Mitchell. *Web Scraping with Python: Collecting Data from the Modern Web*. O'Reilly, 2015.

[7] Margarita Osadchy, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Pérez-Cabo. No Bot Expects the DeepCAPTCHA! Introducing Immutable Adversarial Examples with Applications to CAPTCHA. *IACR Cryptology ePrint Archive*, 2016:336, 2016.

[8] Jeff Yan and Ahmad Salah El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 279–291. IEEE, 2007.

[9] Arun Patala. Using deep learning to break a Captcha system. *Deep Learning*, 2016. https://deepmlblog.wordpress.com/2016/01/03/how-to-break-a-captcha-system/.

[10] Ray Smith. An overview of the Tesseract OCR engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 629–633. IEEE, 2007.

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[12] Patrick Li. Github - captcha_recognize, 2018. https://github.com/PatrickLib/captcha_recognize Commit #19.

[13] Cory LaViska. https://labs.abeautifulsite.net/simple-php-captcha/.

[14] Hsiaoming Yang, 2017. https://pypi.python.org/pypi/captcha/0.2.4.

[15] The Undernet Directory, 2018. http://underdj56iovcytp.onion/page/register.

[16] CoinMixer Anti DoS, 2018. http://coinmixibh45abn7.onion/.

[17] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.

[18] Scrapy Developers. https://scrapy.org/.

[19] Google, 2017. https://developers.google.com/recaptcha/docs/verify.

[20] Kevin Csuka and Dirk Gaastra, 2018. https://gitlab.os3.nl/kcsuka/repo_rp62.

[21] Death by Captcha, 2018. http://www.deathbycaptcha.com.

[22] Patrick Li. Github - tesseract, 2018. https://github.com/tesseract-ocr/tesseract Commit #2,198.

A Hardware specifications of the server used

Machine	Dell PowerEdge R815
Bios-release-date	04/19/2011
Operating System	Ubuntu 16.04.3 LTS
Processors	4 CPUs, each 12 cores, AMD Opteron(tm) Processor 6172
Disk	SCSI Disk, Dell PERC H700, 255GiB (274GB)
Memory	16x 8GiB DIMM DDR3 Synchronous 1333 MHz, Samsung M393B1K70DH0-YH9
Network	NetXtreme II BCM5709 Gigabit Ethernet