The effect of information controls on developers in China: An analysis of censorship in Chinese open source projects

Jeffrey Knockel, Masashi Crete-Nishihata, and Lotus Ruan
The Citizen Lab, Munk School of Global Affairs, University of Toronto
`{jeff,masashi,lotusruan}@citizenlab.ca`

Abstract

Censorship of Internet content in China is understood to operate through a system of intermediary liability whereby service providers are liable for the content on their platforms. Previous work studying censorship has found huge variability in the implementation of censorship across different products even within the same industry segment. In this work we explore the extent to which these censorship features are present in the open source projects of individual developers in China by collecting their blacklists and comparing their similarity. We collect files from a popular online code repository, extract lists of strings, and then classify whether each is a Chinese blacklist. Overall, we found over 1,000 Chinese blacklists comprising over 200,000 unique keywords, representing the largest dataset of Chinese blacklisted keywords to date. We found very little keyword overlap between lists, raising questions as to their origins, as the lists seem too large to have been individually curated, yet the lack of overlap suggests that they have no common source.

1 Introduction

Censorship of Internet content in China is conducted through a system of intermediary liability or "self-discipline" in which service providers are held liable for content on their platforms (MacKinnon, 2011). Service providers are expected to invest in technology and personnel to implement content censorship according to government regulations. Previous work has identified content censorship in a range of applications used in the Chinese market including chat apps (Knockel et al., 2011; Crandall et al., 2013; Hardy, 2013; Ruan et al., 2016), live streaming services (Knockel et al., 2015; Crete-Nishihata et al., 2016), blogs (MacKinnon, 2010), microblogs (Bamman et al., 2012; Miller, 2017), search engines (Villeneuve, 2006), and online games (Knockel et al., 2017). These studies found that the system of censorship in China is decentralized and fragmented due in part to the vaguely defined content regulations and multiple lines of authority between private companies and government bodies. This fragmentation leads to significant variability in how censorship is implemented across different products even within the same industry segment.

While the information control pressures on commercial companies operating in China are well understood, the effect of these controls on individual developers is understudied. Do individual developers include censorship features such as keyword filtering lists in their projects? If they do what is their motivation and how do they create the lists? In this paper we work toward understanding how information controls affect individual developers in China by identifying software projects with keyword blacklists on the popular code repository GitHub. The presence of keyword blacklists used to trigger censorship on GitHub projects suggests that developers perceive or experience some level of informal or formal pressure to include censorship features in their projects.

We developed a novel heuristic tool to extract lists of strings from a variety of structured data formats as used in projects hosted on GitHub. We then designed a method for reliably determining whether a list of strings is a sensitive list of Chinese keywords. These techniques resulted in collection of over 1,000 keyword blacklists extracted from GitHub projects, which collectively contain hundreds of thousands of unique keywords representing the largest dataset of Chinese keyword blacklists collected to date.

We conclude with discussion of questions to investigate in future research including understanding what motivates the developers to include these keyword lists in their projects and how their blacklists are created. Developers may be concerned that they themselves or others using or deploying their projects may fall under the Chinese regulation of service providers and that they may be held liable for content shared on their applications in a similar way to how commercial companies are regulated. They may also be generally accustomed to censorship requirements for software projects and understand it as necessary for introducing an application to the Chinese market. It may also be that they believe their applications should be censored and share the political concerns that motivate the Chinese government. We propose an interview study with the developers to gain insights into their motivations. The process that developers use to create the lists is also an open question. Given the size of these lists, their origin is unclear, as they seem too large for developers to have individually compiled, yet given the lack of overlap between the lists, it seems they are not coming from common sources either. This finding is similar to analysis of censorship keyword lists extracted from commercial applications used in China, which also found lack of overlap between products and industry segments (Crandall et al., 2013; Knockel et al., 2015; Crete-Nishihata et al., 2016; Knockel et al., 2017). Follow-up qualitative research on the developers and analysis of GitHub commit history may provide us with a better understanding of how the lists are created and shared. Overall, the keyword lists we extracted from developer projects and the open questions they present reveal a further avenue to probe the underlying motivations and processes behind application-based censorship in China.

2 Background

Founded in 2008, GitHub is a globally popular open source software development and sharing platform. As of 2017, GitHub reportedly had 24 million developers working across 67 million repositories (GitHub, 2017). In 2017 alone, 692,000 users from China signed up for an account on GitHub, and nearly one-third of all China-based developers used GitHub (崔绮雯, 2015), many of whom worked for Chinese Internet giants including Baidu, Tencent, and Alibaba (Open Source China, 2016).

GitHub has also been the target of censorship in China. On January 21, 2013, GitHub was blocked in China using DNS hijacking (Protalinski, 2013). While there was no official response from the Chinese government, Chinese users speculated that the blocking was due to GitHub's hosting of plug-in software that allowed purchasing of train tickets before the Spring Festival rush in China, which was discouraged by the Ministry of Railways (雅楠, 2013). On January 23, 2013, GitHub was accessible again in China (Bai Tiantian, 2013).

On January 26, 2013, GitHub's China-based users experienced a MITM attack in which attackers could have intercepted traffic between the site and its users in China. (GreatFire, 2013) The mechanism of the attack was through a fake SSL certificate. The motivations behind this attack were not clear, but it may have been used to collect the passwords of China-based GitHub users. The attack was lifted after about an hour.

On March 26, 2015, the Chinese censorship apparatus employed a new tool, the "Great Cannon", to engineer a distributed-denial-of-service (DDoS) attack on two GitHub pages run by GreatFire.org, an advocacy organization dedicated to documenting and circumventing Internet censorship in China and cn-nytimes, a Chinese language version of The New York Times, launching the largest DDoS attack in the site's history as of 2015 (Marczak et al., 2015). However, despite the often adversarial relationship between GitHub and the Chinese government, China-based developers use the site because, unlike many platforms such as Facebook, Twitter, or Google, there is no commonly-used Chinese alternative to GitHub (Claburn, 2013).

While the technology industry in China is huge, service operators face unique challenges due to the country's strict regulatory environment. Any service provider operating in the Chinese market is required to comply with government content regulations. Commercial companies are held responsible for content on their platforms and are expected to dedicate resources to ensure compliance with relevant laws and regulations. Failure to do so can lead to fines or revocation of business licenses (MacKinnon, 2009).

This system is a form of intermediary liability or "self-discipline", which allows the government to shift responsibility for information control to the private sector (MacKinnon, 2010). While the response of commercial companies to regulations in China is well researched, the response of individuals providing Internet services is understudied, despite individuals also falling under many of these regulations (Ministry of Industry and Information Technology, 2005; OpenNet Initiative, 2006).

Laws and regulations on content control in China on content are broadly defined. In 2010, the State Council Information Office published a list of prohibited topics that are vaguely defined, such as "spreading rumors, disrupting social order and stability" and "transgressing social morality" (State Council Information Office and Ministry of Information Industry, 2005).

In 2017, the Cyberspace Administration of China (CAC), China's top-level Internet governance body, released four major regulations on Internet management, ranging from strengthening real-name registration requirements on Internet forums and online comments, to making individuals who host public accounts and moderate chat groups liable for content on the platforms (Cyberspace Administration of China, 2017a; Cyberspace Administration of China, 2017b; Cyberspace Administration of China, 2017c; Cyberspace Administration of China, 2017d). These regulations further push down the responsibility of content control down to individuals, a change which David Bandurski describes as "atomization and personalization of censorship" in China (Bandurski, 2017).

3 Methodology

In this section we describe (1) how we collect files, (2) how we extract lists of strings from these files, and (3) how we determine which of these lists are lists of sensitive blacklisted Chinese keywords.

3.1 Collecting files from GitHub

To collect files potentially containing Chinese keyword blacklists, we searched the popular code repository GitHub for keyword blacklists contained in files on the site. The files we searched were largely part of open source projects, but our search included any file publicly uploaded onto the site.

To search the site, we scraped the GitHub search function on their website. This step was necessary as, although GitHub has an API, it does not include full text search of all files across their entire site. Scraping websites can have ethical implications. In this case, scraping GitHub's web search was not allowed by their `robots.txt` file. Moreover, we did not want to deny service to any GitHub users by repeatedly making expensive search queries. To account for these concerns, we contacted GitHub's support team and communicated to them our desire to scrape their web search functionality. They permitted us to scrape their site, asking for the IP addresses of our measurement machines and the HTTP user agent string identifying our scraper. Moreover, they requested that our scraper not exceed ten HTTP requests per minute, as performing full site code search was computationally expensive. We wrote a scraper to automatically perform web searches, following up to the tenth page of results, returning at most 100 results, and complying with GitHub support's suggested rate limit.

To design our search queries, we first compiled a list of sensitive search words from publicly available Chinese blacklists reverse engineered from chat ("knowhow", 2004; Knockel et al., 2011; Crandall et al., 2013; Hardy, 2013) and live streaming (Knockel et al., 2015) apps. The former included QQ Games, TOM-Skype, Sina UC, and LINE. The latter included YY, 9158, Sina Show, and GuaGua. Together, we had over 21,934 unique keywords. As many of these keywords may be commonly mentioned in files not containing a sensitive keyword blacklist, we did not want to search for each of these keywords individually. Instead, we combined each sensitive keyword with each of the following blacklist-related strings to be found in either the file's name or contents: "badword", "banned", "blacklist", "censor", "dirty", "filter", "forbid", "forbidden", "illegal", "keyword", "profanity", or "sensitive". These strings were largely inspired by the names of files in previous work and by words we *a priori* speculated might be used.

As an optimization to reduce the number of HTTP requests necessary for each search, we first searched for each sensitive keyword by itself before combining it with each blacklist-related string. If the number of pages of results is no more than twelve, the number of blacklist-related strings we would combine it with, then we download all of the pages and manually filter out results that do not include blacklist-related strings in the files or file names. This step reduces the number of requests for such sensitive keywords as we do not have to make a separate request for each blacklist-related string. This strategy is especially more efficient for sensitive keywords that have no results or only one page of results, as only one request is necessary for such keywords as opposed to twelve.

We started scraping June 26 2016 and finished July 26 2016 after completing all search queries. For each of our search results, we downloaded each file corresponding to a search result in a separate stage of our process. We then converted each file to UTF-8 by using the `chardet` python package to detect each file's encoding, decoding it, and then re-encoding to UTF-8. Since we found that `chardet` often misclassified files as GB2312 or GBK containing GB18030 characters, we decoded all files classified as GB2312 and GBK as GB18030. If a file was encoded with a single byte encoding such as ASCII or Latin-1, we discarded the file as such an encoding does not have the ability to encode Chinese characters.

3.2 Extracting lists of strings from files

After downloading all files potentially containing Chinese blacklists and reencoding them to UTF-8, we extracted lists of strings from each file using a script we wrote called the *list extractor*. We used a heuristics approach to support a large variety of file formats. Our heuristics were based on (1) *a priori* assumptions about what file formats would be commonly used to store blacklisted keywords, (2) insights from previous work that reverse engineered keywords from applications (Crandall et al., 2013; Knockel et al., 2015; Knockel et al., 2017), and (3) different file formats that we found used to store blacklisted keywords in the files we downloaded from GitHub containing the word "法轮" (Falun), as Falun Gong is a taboo Chinese religious group referenced in all Chinese blacklists discovered in previous work (Crandall et al., 2013; Knockel et al., 2015; Knockel et al., 2017)

As a design goal we sought to keep the list extractor as simple as possible, and we were conscious to avoid overfitting to cases that seemed small or non-representative. Our list extractor was ultimately implemented in 348 lines of python3 source code, including comments and using typical whitespace conventions. It ultimately supported the following *formats* of files: XML, JSON, CSV, delimited plain text, C-like code, and newline-delimited plaintext. Our string extractor attempts to extract lists of strings from each file according to the formats listed in the above order. As a rule, we require that all lists have at least 20 strings and that they contain at least one Chinese character. If such a list is found using a format, then no other formats are attempted. In the remainder of this section, we outline our implementation of each format.

We first attempt to interpret each file as XML and parse it into a tree using a standard parser provided by python. If the file successfully parses, we then *flatten* the resulting tree by turning it into a list of pairs (x, y) where, for each attribute value or text node x, its path y consists of the names of all parent elements of x from the root node downward and, in the case of an attribute value, the name of the attribute key. We then consider all strings x with the same path y to be strings in the same list.

For XML, we also found that lists would often be contained within the text of a single XML attribute value or text node separated by a delimiter. Thus, we test each attribute value and text node to see if it contains a *delimited list*. A delimited list is a string containing delimiters, where a delimiter is one of the characters "`, |~;_@`", delimiting substrings that are no longer than 100 characters. Also, it must satisfy the requirements of any of our lists, namely that it contains at least 20 strings and that at least one of them contains a Chinese character.

Next, we attempt to parse the file as JSON using python's JSON parser. If the file successfully parses, we perform a flattening procedure to extract lists of strings similar to the one we do for XML, except building paths out of recursive JSON arrays and lists. Similar to XML, we check each string in the JSON to see if it contains a delimited list. Finally, as JSON inherently supports list-like data structures, we treat any JSON list where all elements are strings as a list of strings. Similarly, if any JSON array's keys are all strings or its values are all strings, then we treat those keys or values, respectively, as a list of strings.

After JSON, we attempt to parse the file as a CSV file, although we generally do not require that commas act as the separators. Although python includes support for parsing CSV files, we found the parser inadequate for our purposes, as it assumes that the input is a CSV file and is therefore quite liberal in what input it accepts. Contrary to this, we did not know if the file we were parsing was a CSV file, and so we wanted the parsing to consistently fail if the file was not a CSV.

To parse a CSV file, for each possible separator "`;,|^\t `" and for each possible quote symbol "`"'~`", we attempt to parse a first row of the CSV according to these separators and quote symbols and determine how many columns exist in the first row. If there are between two and ten columns, inclusive, we then attempt to parse the remainder of the file, according to the same separators and quote symbols and using the same number of columns as in the first row. If the parser finds that any row has a different number of columns, then the parsing of the file with that separator and quote symbol fails, and we then try any remaining unattempted combinations of separators and quote symbols. Lines containing only whitespace are ignored. If a file successfully parses as a CSV file, then, if it contains at least 20 rows, each column containing a Chinese character is considered a list.

Next, we attempt to parse the entire file itself as a plain delimited list, containing no other structured data.

After this, we attempt to treat the file as a C-like language. First, we remove all C- and C++-style comments from the file. Then we attempt to search it for arrays (`{…,…}`), lists (`[…,…]`), and tuples (`(…,…)`) of string literals (`"…\"…"` or `'…\'…'`), as well as delimited lists inside single string literals and regular expression literals such as those in Javascript (`/…\/…/`).

Finally, we attempt to treat the file as a plain newline-delimited list of strings. To distinguish such a file from any file containing multiple lines, we place some constraints on what files we accept based on what a Chinese blacklist would likely look like. First, if any line is not entirely whitespace and begins with at least three spaces, we reject the file, as indentation is a strong indication that the file contains structured data. Second, if the average length of all lines exceeds 15 characters, then we reject the file. Otherwise, if the file meets the usual list conditions, namely being at least 20 lines and containing a Chinese character, we accept it as a list of strings.

Another desirable format to have implemented would have been SQL. However, we found extracting lists from SQL problematic. The grammar of the language itself is not consistent between databases, and SQL allows diverse ways to insert list-like data into a database. With effort, we suspect that SQL could have been supported, but such an implementation would have been at odds with the design goals of keeping the string extractor simple and not overfitting to specific examples of data.

3.3 Determining whether lists of strings are Chinese blacklists

After we have extracted any potential lists of strings from each file, our final step is to classify each list as a *Chinese blacklist* or not. Our goal at this stage is to remove any non-blacklist-related lists of strings such as a list of Chinese cities including any list of strings extraneously extracted by our string extractor.

For purposes of this work, we consider a Chinese blacklist to be a list of sensitive keywords suitable for triggering censorship or surveillance that is primarily targeted at a Chinese audience. As a counterexample, we found that some projects in GitHub maintain lists of profane words from a large number of languages, including Chinese. In this work, we were not interested in detecting these lists, as they were not specifically targeting Chinese users and generally not made to comply with Chinese law.

To classify lists, we evaluate two approaches, (1) a *naive approach* based on searching for a single substring and (2) a more sophisticated *machine learning approach*. The naive approach works by positively classifying all lists that have any string containing "法轮" (Falun), referring to Falun Gong. We then manually validate each positively classified list to verify that it is indeed a Chinese blacklist. To extend the results, for each list containing "法轮", we additionally manually inspect that list's file for any other lists that may be Chinese blacklists and add any to the Chinese blacklists we had already found and verified. While this approach is simple, the weakness of it, however, is that it can only identify Chinese blacklists that either contain "法轮" or are in a file with a list containing "法轮".

To overcome this limitation, we also used a machine learning approach. We used a one-class support vector machine (SVM) (Schölkopf et al., 2001), as implemented by Scikit-learn (Pedregosa et al., 2011) and LIBSVM (Chang and Lin, 2011). We chose this machine learning classifier because it had the desired

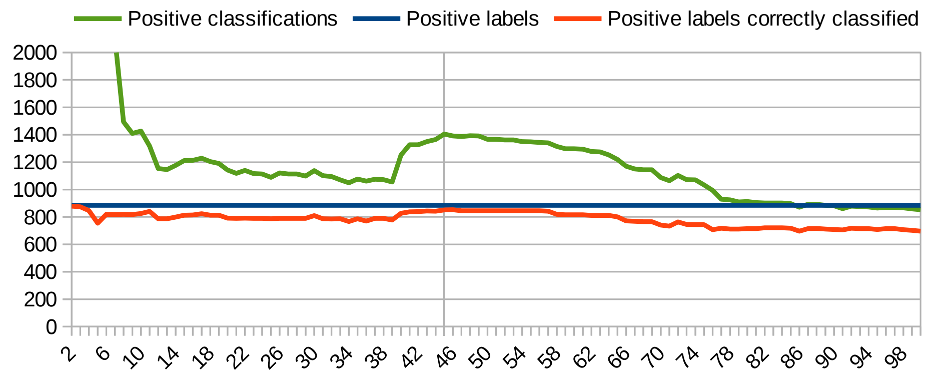

Figure 1: Classification performance for different d, the number of dimensions.

property of requiring training using only one class, the positive class, while still being able to classify something as either being in that class or not. This type of classifier was desirable because, although we had previous examples of Chinese blacklists to use to train a classifier, we had no representative sampling of all lists of strings on GitHub that are not Chinese blacklists.

The size of our training set was limited by the number of lists publicly available at the time of our work. Our training data consisted of the publicly available Chinese blacklists we used to design our search queries. We additionally used a list discovered in Google's Javascript code used to filter their mainland Chinese search engine ("caiguanhao", 2012). Overall, our training set consisted of 27 different Chinese blacklists.

In our model, we consider each list to be a vector of counts of the number of occurrences of each *Chinese word* in that list. We consider a Chinese word to be every consecutive string of Chinese characters in any of the strings in that list. For example, "历史de伤口" (history of wounds) contains two Chinese words, "历史" (history) and "伤口" (wound), separated by "de". As a preprocessing step, we also convert all traditional Chinese characters to simplified characters.

To reduce the dimensionality of this dataset, we use singular-value decomposition. Since we had all unlabeled data available at the time of training, we reduced dimensions over both the training data and the unlabeled data as a whole. Thus, in addition to making the classification computationally tractable, this allowed us to incorporate our unlabeled data into our training insofar as it is used to determine the final dimensions.

Since it was not obvious what number of dimensions to reduce our dataset to, we tried each number of dimensions d from 2 to 100. Cross-validating a one-class classifier is not generally feasible as there is no negative-labeled data to use for cross-validation. As a further difficulty, our training set was comparatively small compared to the number of lists that we could potentially detect.

To evaluate the best value of d, we used the verified Chinese blacklists discovered using our naive approach as positive labels for purposes of evaluating each d. We generally used the value of d that maximized the number of correctly classified positive labels. As with our naive approach, with the machine-learning approach we also manually verified each positive classification. Thus, we were not concerned with small increases to the false positive rate by maximizing the number of positively classified blacklists as long as it did not prohibitively burden our task of manually verifying positives by creating too many false positives.

4 Results

By scraping GitHub, we downloaded a total of 648,323 files. From these, using our list extractor, we extracted 45,986 lists containing at least 20 strings and at least one Chinese character.

Using our naive classification technique, we selected those lists with a string containing "法轮", yielding 915 potential Chinese blacklists. After manually verifying each list, we reduced this number to 838 true Chinese blacklists. By manually inspecting the other lists in the files containing these Chinese blacklists, we found an additional 46 Chinese blacklists, yielding a total of 884. Accounting for duplicates, we

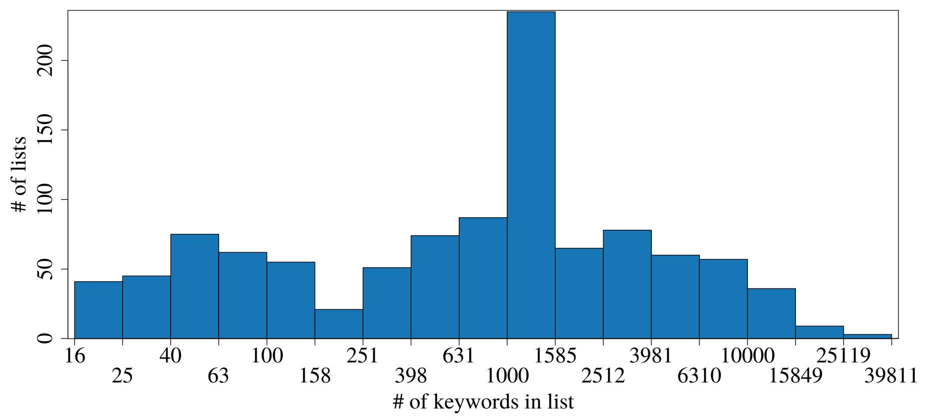

Figure 2: The distribution of the number of keywords in each list.

Top 1–10			Top 11–20		
n	Keyword	Translation	n	Keyword	Translation
703	鸡巴	dick	621	台独	Taiwanese independence
689	法轮	Falun	620	阴唇	labia
665	李洪志	Li Hongzhi	618	真善忍	truthfulness, tolerance
640	阴道	vagina	616	疆独	Xinjiang independence
638	阴茎	penis	616	做爱	making love
635	藏独	Tibetan independence	611	口交	blowjob
633	龟头	glans	604	法轮功	Falun Gong
629	淫水	kinky	597	性交	sex
626	肛交	anal sex	596	共匪	CCP bandit
622	小穴	small hole	593	江泽民	Jiang Zemin

Table 1: Top 20 keywords as ranked by n, the number of lists each keyword appears in.

overall found 451 unique Chinese blacklists using this technique.

Using our machine learning technique, in Figure 1, for each number of dimensions d, we show the number of positive classifications and the number of positive labels correctly classified. For $d > 3$, the maximum number of positive labels correctly classified, 852, occurs at $d = 46$. For degenerate values of d, $d = 2$ and $d = 3$, more of the positive labels were positively classified; however, for both of these d, the number of positively classified lists is over 20,000, which is nearly half of our unlabeled dataset.

Using $d = 46$, our machine learning approach positively classified 1,391 lists. After manually verifying them, we found that 1,054 of the positive classifications were true Chinese blacklists. Accounting for duplicates, there were 524 unique Chinese blacklists. Together these lists comprised 215,007 unique keywords. In the remainder of this section, we characterize these 1,054 true Chinese blacklists that we collected using the machine learning approach.

The Chinese blacklists we discovered contained a varying number of keywords. Since our methodology only considered lists with at least 20 keywords, all lists that we found had at least that many. The longest blacklist we found contains 38,237 keywords. The mean length was approximately 2,128 keywords, and the median length was 1,026 keywords. See Figure 2 for a plot of the distribution.

The blacklists contained a variety of different sensitive keywords. Table 1 shows the top 20 keywords most commonly occurring on the lists. These keywords reflected prurient interests (鸡巴, “dick”), Falun Gong references (法轮, “Falun”; 李洪志, “Li Hongzhi”, the founder of Falun Gong), political movements (藏独, “Tibetan independence”), government criticism (共匪, “CCP bandit”), and political leaders (江泽民, “Jiang Zemin”).

Despite a small number of words being on most lists, we found that blacklists were typically very dis-

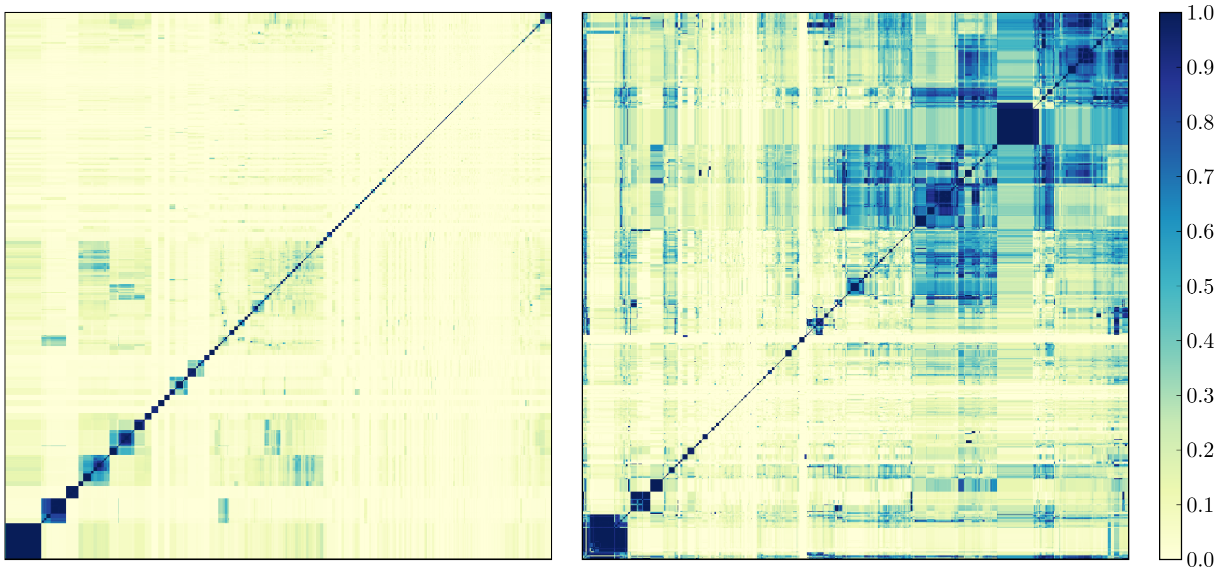

Figure 3: Left, lists clustered by Jaccard similarity; right, lists clustered by *similarity*(x, y) = max(% of x in y, % of y in x).

similar. We compared the inter-similarity between blacklists using two metrics. The first metric compares their Jaccard similarity, which is the size of the set intersection of both lists divided by their set union. The second metric, for two lists x and y, computes max(% of x in y, % of y in x). This metric is designed to tease out relationships where one list was built using another; however, using this metric, small lists which typically contain mostly commonly blacklisted words tend to be very similar to large lists that tend to inevitably contain them as well. Figure 3 shows the lists compared using each metric and clustered according to the centroid linkage method (Müllner, 2011).

The heat maps reveal very little overlap between lists. This finding is consistent with studies analyzing the implementation of censorship on Chinese chat clients, live streaming apps, and mobile games (Crandall et al., 2013; Knockel et al., 2015; Knockel et al., 2017) by commercial companies. These studies found that, rather than Chinese censorship being top-down and monolithic, it is a decentralized system where developers are liable for deciding what to censor themselves. The unprecedented quantity of lists in our study raises new questions regarding how so many disparate lists are being created, especially since they are so large, with over half of them having over one thousand keywords.

5 Conclusion and future work

In this work we scraped GitHub for files containing sensitive Chinese keywords and strings related to blacklists. From these files, we used a novel heuristics-based tool to extract lists of strings from a number of structured data formats. We then used a machine learning technique to determine which of these lists represent Chinese blacklists. Overall, we found over 1,000 Chinese blacklists comprising over 200,000 unique keywords, representing the largest dataset of Chinese keyword blacklists collected to date. These results provide multiple avenues for future research particularly around what motivates individual developers to include these lists in their projects and how the lists are created.

It remains an open question why developers include these lists in their projects. The developers may be concerned that they themselves or others using or deploying their projects may be held liable for content shared on their projects in the same manner that commercial companies are known to be controlled. Developers may be accustomed to this requirement and see it as necessary for their project to gain users and traction in the Chinese market. It may also be that they believe their application should be censored and share the political concerns that motivate the Chinese government.

Most of the lists we discovered had over 1,000 keywords, which seems too large for a developer to individually compile, yet given the dissimilarity between lists, they do not appear to come from common sources either. GitHub projects include contact information making it possible to consider an interview of the developers to gain insight into their motivations and process for creating the lists. We found little similarity between lists in terms of specific keywords, suggesting that these lists were independently curated. Categorization of the keywords according to high level topic would allow testing whether lists have similarity when comparing their high level topic coverage.

Another avenue of future work is to further utilize data in GitHub to understand Chinese censorship. In this work we sampled projects containing Chinese blacklists; however, another research direction is to sample all China-based projects independently of if they perform censorship. This approach would allow us to characterize what types and what proportion of China-based projects on GitHub perform filtering versus those which do not. GitHub also contains valuable metadata unexplored by this work. Analyzing blacklists' git commit history may allow researchers to if and how blacklists are updated over time. Moreover, GitHub social networking data tracking forks, stars, and watchers may provide insight into if and how lists are shared over the platform.

Acknowledgments

This research was supported by the Open Society Foundations. Jeffrey Knockel's research for this project was supported by the Open Technology Fund's Information Control Fellowship Program.

Keyword list data is available at `https://github.com/citizenlab/chat-censorship/tree/master/open-source`.

References

Bai Tiantian. 2013. China's GitHub Censorship Dilemma. Available at `http://www.globaltimes.cn/content/757868.shtml`.

David Bamman, Brendan O'Connor, and Noah A. Smith. 2012. Censorship and deletion practices in Chinese social media. *First Monday*, 17(3).

David Bandurski. 2017. The Great Hive of Propaganda. Available at `http://chinamediaproject.org/2017/09/16/the-great-hive-of-propaganda/`.

"caiguanhao". 2012. Google收集的GFW屏蔽关键词（敏感词）. Available at `https://caiguanhao.wordpress.com/2012/06/01/google-gfw-blacklist/`.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Thomas Claburn. 2013. China's GitHub Censorship Dilemma. Available at `https://www.informationweek.com/software/social/chinas-GitHub-censorship-dilemma/d/d-id/1108436`.

Jedidiah R. Crandall, Masashi Crete-Nishihata, Jeffrey Knockel, Sarah McKune, Adam Senft, Diana Tseng, and Greg Wiseman. 2013. Chat program censorship and surveillance in China: Tracking TOM-Skype and Sina UC. *First Monday*, 18(7), 6.

Masashi Crete-Nishihata, Andrew Hilts, Jeffrey Knockel, Jason Q. Ng, Lotus Ruan, and Greg Wiseman. 2016. Harmonized Histories? A year of fragmented censorship across Chinese live streaming applications. Technical report, Citizen Lab, University of Toronto. Available at `https://netalert.me/assets/harmonized-histories/harmonized-histories.pdf`.

Cyberspace Administration of China. 2017a. 互联网用户公众账号信息服务管理规定. Available at `http://www.cac.gov.cn/2017-09/07/c_1121624269.htm`.

Cyberspace Administration of China. 2017b. 互联网群组信息服务管理规定. Available at `http://www.cac.gov.cn/2017-09/07/c_1121623889.htm`.

Cyberspace Administration of China. 2017c. 互联网论坛社区服务管理规定. Available at `http://www.cac.gov.cn/2017-08/25/c_1121541921.htm`.

Cyberspace Administration of China. 2017d. 互联网跟帖评论服务管理规定. Available at http://www.cac.gov.cn/2017-08/25/c_1121541842.htm.

GitHub. 2017. GitHub Octoverse 2017. Available at https://octoverse.github.com/.

GreatFire. 2013. China, GitHub and the man-in-the-middle. Available at https://en.greatfire.org/blog/2013/jan/china-github-and-man-middle.

Seth Hardy. 2013. Asia Chats: Investigating Regionally-based Keyword Censorship in LINE. Technical report, Citizen Lab, University of Toronto. Available at https://citizenlab.ca/2013/11/asia-chats-investigating-regionally-based-keyword-censorship-line/.

Jeffrey Knockel, Jedidiah R. Crandall, and Jared Saia. 2011. Three researchers, five conjectures: An empirical analysis of TOM-Skype censorship and surveillance. In *FOCI'11 (USENIX Workshop on Free and Open Communications on the Internet)*.

Jeffrey Knockel, Masashi Crete-Nishihata, Jason Q. Ng, Adam Senft, and Jedidiah R. Crandall. 2015. Every Rose Has Its Thorn: Censorship and Surveillance on Social Video Platforms in China. In *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*.

Jeffrey Knockel, Lotus Ruan, and Masashi Crete-Nishihata. 2017. Measuring Decentralization of Chinese Keyword Censorship via Mobile Games. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*.

"knowhow". 2004. QQ过滤词列表 zt. Available at https://web.archive.org/web/20040908030852/http://bbs.omnitalk.org/arts/messages/3824.html.

Rebecca MacKinnon. 2009. China's Censorship 2.0: How companies censor bloggers. *First Monday; Volume 14, Number 2 - 2 February 2009*.

Rebecca MacKinnon. 2010. Google's China troubles continue; Congress examines U.S. investment in Chinese censorship. Available at http://rconversation.blogs.com/rconversation/2010/06/index.html.

Rebecca MacKinnon. 2011. China's "Networked Authoritarianism". *Journal of Democracy*, 22(2):32–46.

Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ronald Deibert, and Vern Paxson. 2015. China's Great Cannon. Technical report, Citizen Lab, University of Toronto. Available at https://citizenlab.org/2015/04/chinas-great-cannon/.

Blake Miller. 2017. The Limits of Commercialized Censorship in China. Available at http://www.blakeapm.com/research/censorship.

Ministry of Industry and Information Technology. 2005. 非经营性互联网信息服务备案管理办法. Available at http://www.miit.gov.cn/n1146295/n1146557/n1146624/c3554618/content.html.

Daniel Müllner. 2011. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*.

Open Source China. 2016. GitHub 中国区前 100 名到底是什么样的人？. Available at https://www.oschina.net/news/72235/github-china-100.

OpenNet Initiative. 2006. 非经营性互联网信息服务备案管理办法. Available at https://opennet.net/bulletins/011/.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Emil Protalinski. 2013. The Chinese government appears to be blocking GitHub via DNS (Update: Investigation underway). Available at https://thenextweb.com/asia/2013/01/21/the-chinese-government-appears-to-have-completely-blocked-github-via-dns/.

Lotus Ruan, Jeffrey Knockel, Jason Q. Ng, and Masashi Crete-Nishihata. 2016. One App, Two Systems: How WeChat uses one censorship policy in China and another internationally. Technical report, Citizen Lab, University of Toronto. Available at https://citizenlab.ca/2016/11/wechat-china-censorship-one-app-two-systems/.

Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471.

State Council Information Office and Ministry of Information Industry. 2005. Provisions on the Administration of Internet News Information Services. Available at `http://www.cecc.gov/pages/virtualAcad/index.phpd?showsingle=24396`.

Nart Villeneuve. 2006. The filtering matrix: integrated mechanisms of information control and the demarcation of borders in cyberspace. *First Monday*, 11(1).

崔绮雯. 2015. 已经被攻击了一百多小时的GitHub为什么那么重要? . Available at `https://www.qdaily.com/articles/7938.html`.

雅楠. 2013. 12306抢票插件拖垮美国代码托管站Github. Available at `http://tech.sina.com.cn/i/csj/2013-01-16/19367984516.shtml`.