Open-Source License Violations of Binary Software at Large Scale

Muyue Feng*†, Weixuan Mao‡, Zimu Yuan*, Yang Xiao*†, Gu Ban*†, Wei Wang*, Shiyang Wang*†,
Qian Tang*†, Jiahuan Xu*, He Su*†, Binghong Liu*, Wei Huo*†
*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{fengmuyue, yuanzimu, xiaoyang, bangu, wwei, wangshiyang, tangqian,
xujiahuan, suhe, liubinghong, huowei}@iie.ac.cn
†School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡National Computer Network Emergency Response Technical Team / Coordination Center of China, Beijing, China
maoweixuan@cert.org.cn

***Abstract*—Open-source licenses are widely used in open-source projects. However, developers using or modifying the source code of open-source projects do not always strictly follow the licenses. GPL and AGPL, two of the most popular copyleft licenses, are most likely to be violated, because they require developers to open-source the entire project if any code under GPL/AGPL protection is included whether modified or not. There are few license violation detectors focusing on binary software, owning to the challenge of mapping binary code to source code efficiently and accurately at large scale. In this paper, we propose a scalable and fully-automated system to check open-source license violation of binary software at large scale. We match source code to binary code by analyzing file attributes of executable files and code features that are not affected by compilation and could vary between projects. Moreover, to break the barrier of large-scale analysis, we introduce an automatic extractor to parse executable files from installation packages that are broadly available in software download sites. In empirical experiments of binary-to-source mapping, we have got a remarkable high accuracy of 99.5% and recall of 95.6% without significant loss of precision. Besides, 2270 pairs of binary-to-source mapping relationships are discovered, with 110 license violations of GPL and AGPL licenses related to 7.2% of the 1000 real-world binary software projects.**

I. Introduction

Nowadays, more and more open-source projects are protected by software licenses in copyright. Nearly 1 million new projects created last year in Github were under open-source licenses [1]. These open-source projects, especially third-party libraries, are widely reused by private and commercial software projects. However, the copyright of open-source project is not well protected as it should be.

There are thousands of open-source licenses with different restrictions on concrete copyrights. But GNU General Public License (GPL) and Affero General Public License (AGPL), which are the representatives of copyleft licenses, occupied 35% distribution of all open-source licenses in total [2]. They require users to open the source code of the whole projects if some code under their protection are included whether modified or not. It is not hard to understand why GPL and AGPL are most likely to be violated. Nevertheless, there are only a few studies on how to check the violations automatically.

[3][4][5] propose several methods of automatically detecting license violations or inconsistencies of open-source software programs. And [6][7] implement tools targeting Android native code and embedded firmwares. However, binary software programs are still the majority in personal computer, especially in Windows platform. And there is neither automatic license violation detection system for them nor study at large scale.

The core challenge is mapping binary code of binary software programs to source code of open-source projects under the great affection of compilation. Besides, executable files of software programs can not always be obtained directly, because most software programs are released as installation packages.

To address these issues, we propose a scalable and fully-automated violation checking system of binary software targeting GPL/AGPL at large scale. To do binary-to-source mapping, we extract file attributes (e.g., company name, legal copyright) and code features for both binary software and open-source projects. The code features we used are not affected by compilation and could vary between projects. To get executable files at large scale, we introduce an automatic extractor to parse executables from installation packages. The extractor can unpack installation packages by corresponded unpackers and compressors, and simulate the interaction between users and programs to install softwares if they are packed by customized packers or download executables from Internet while installing.

The experiments show that our system is capable to efficiently map thousands of binary software programs to hundreds of open-source projects, costing 56.5 seconds for each binary-to-source matching on average. Project name labeling based on file attribute contributes 26% of binary-to-source mapping results within minutes. And the code feature matching module achieves an accuracy of 99.5%, a remarkably high recall of 95.6% and a precision of 82.7% that has no significant loss. We have found 2270 pairs of binary-to-source mapping while comparing 1000 binary software programs and 264 commonly-used open-source projects. After manually verification of licenses of the top 100 frequently reused projects, we find 9 projects that use GPL/AGPL only and 110 license violations related to 53 binary software programs. In short, there are 7.2% of our experimental binary software programs violate open-source licenses.

II. Background and Related Work

A. Background of Licenses

There are thousands of open-source licenses, but the top 10 commonly-used licenses contribute more than 94% of the distribution [2]. These licenses can be divided into two categories, permissive licenses and copyleft licenses (Table I).

978-1-7281-0591-8/19/$31.00 © 2019 IEEE

TABLE I
CLASSIFICATION OF LICENSES

Classification		Open-Source if only Use it	Open-Source if Use and Modify it
Permissive Licenses	BSD	×	×
	MIT	×	×
	Apache	×	×
Copyleft Licenses	LGPL	×	✓
	Mozilla	×	✓
	GPL	✓	✓
	AGPL	✓	✓

A permissive license, like BSD license, MIT license and Apache license, requires users to declare the original author's copyright when the open-source code is used. But a copyleft license, like GNU General Public License (GPL), Affero General Public License (AGPL), GNU Lesser General Public License (LGPL) and Mozilla Public License, requires users to open source their code in some cases.

Copyleft licenses can also be divided into two groups. The more restrictive group, containing GPL and AGPL, asks users to open source the whole project only if the code is included whether modified or not. In contrast, the less restrictive group claims that users should open source their code which is modified from the open-source project.

The more restrictive group of copyleft licenses, mostly GPL and AGPL, is the most often violated ones because of their strong restriction of using. GPL and AGPL have occupied 35% distribution of all open-source licenses [2], but there are only a few studies on them.

B. Related Work

To implement an open-source license violation detector, there are two barriers needed to be broken: 1) to recognize the origin of code, and 2) to detect violating behavior of target projects. Detectors can be divided into two groups. One group targets to open-source projects and the other targets to binary software projects. The difficulty of identifying the origin of code is obviously different between them.

For open-source projects, it is much easier to recognize the origin of source code. [4] identifies code origin by textual license terms in the head of source files. [3] directly uses existing tools to calculate the similarity between code. [8][9] check license validation by parsing Software Package Data Exchange (SPDX) files. For binary programs, [6] maps binary code to source code targeting Android native code, and [7] implements a tool aiming at firmwares. However, there is no mature system being capable to do binary-to-source mapping targeting complex desktop software programs (e.g., PE and ELF files) efficiently and accurately at large scale.

III. OPEN-SOURCE LICENSE VIOLATION CHECK

A. Challenges

Open-source license violation detection has three important steps: 1) recognize the open-source projects included by binary code, 2) collect the license information of open-source projects, and 3) confirm whether the usage condition matches the terms of open-source licenses. The first step is the bottomneck that needs to be broken through.

Open-sourced code exists in binary software in two way, directly used as a single library file and packaged by customized code in a library file or an executable program. In the former case, the original project name and copyright declaration might be retained. However, for Windows Portable Executable (PE), the names and attribute information of library files are easy to be removed or modified. Therefore, file names and attributes are not reliable. Code features are much credible.

In existing research of binary-to-binary comparison, instructions [10] and control flow graphs [11] [12] are the most commonly-used code features. But under the circumstances of binary-to-source mapping, all these features have obvious limitations. The two objects to be compared are no longer the same type of code. One is machine code and the other is high-level language.

String features are easy to remove, especially as log messages and error messages. Integer constants exists with noises, because enormous integers regardless to source code will be included into binary code during compilation, e.g., memory offsets. Control flows are greatly influenced by compilation and optimization operations, such as function inlining and code block merging or splitting, stack protection and so on. Thus, there is still no effective way to establish code relationships between binary code and source code.

TABLE II
EXAMPLES OF PE FILE ATTRIBUTES

Feature Name	Attribute Value	Regularized Value
File Name	swscale-4.dll	swscale
Internal Name	swscale-3.dll	swscale
Company Name	FFmpeg Project	FFmpeg
Legal Copyright	Copyright (C) 2000-2015 FFmpeg	FFmpeg

TABLE III
CODE FEATURES USED IN BINARY-TO-BINARY MAPPING AND BINARY-TO-SOURCE MAPPING (OUR SYSTEM)

Feature Name	Bin-to-Bin	Bin-to-Src
Export	✓	✓
String	✓	✓
Integer Number	✓	
String Array		✓
Global Number Array		✓
Global Enum Array		✓
CFG	✓	
CG	✓	
Switch/Case		✓
If/Else		✓

B. File Attributes

File attributes, including file name, internal name, product name, company name, legal copyright, etc., are important features for identifying original source of code in PE files. They can be extracted from the VERSIONINFO resource of PE files. These features may come from the original developers of open-source projects, as well as secondary releasers of them.

Table II lists examples of file attributes that are used in our system. To uniform file attributes with similar information but

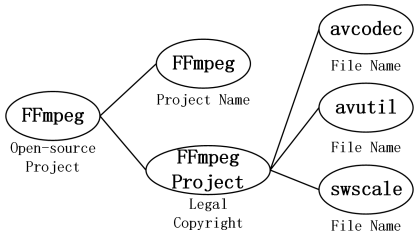

Fig. 1. Relationship Propagating of FFmpeg

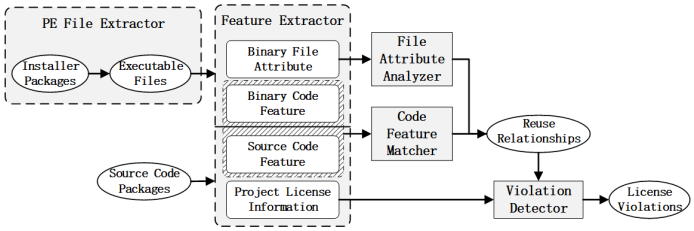

Fig. 2. System Architecture

subtle textual differences, we regularize them by eliminating repeated spaces, useless punctuations and meaningless common words, e.g., "copyright" and "2015".

Then, to identify the original file attributes from the open-source library official release rather than secondary release, we design some heuristic rules for different types of file attributes based on over 230K PE files we have accumulated.

For example, to discover relationships between legal copyrights and file names, we count for the co-occurence frequency of file names with a specific copyright string. If the quantity of file names is less than 1/10 of the frequency of the copyright string, which means the binary files with this copyright string have the same group of file names repeatedly, the copyright string is very likely to be a specific feature of a third-party library, and these file names are strongly related to this library. That is how the relationships between “FFmpeg Project” and "avcodec"/"avutil"/"swscale" are established as shown in Fig. 1.

Finally, we build the attribute relationship graph on open-source libraries and their attributes by establishing the edge of library-to-attribute relationship and attribute-to-attribute relationship, and propagating the relationships by edge neighbors.

C. Code Feature

To choose code features suitable with binary-to-source cases, we evaluate each feature by two dimensions, whether it would be affected by compilation and could vary between projects. There are 7 kinds of features perform well in both dimensions as shown in Table III, including 2 kinds of normal features (including export functions and strings), 3 kinds of constant data features and 2 kinds of control flow features.

1) Constant Data Features: Although integer constants are frequently used as feature of binary code in state-of-the-art researches, they are still obviously affected by compilation. While compilation and optimization, lots of immediate numbers regardless to source code, such as memory offsets and stack offsets, are introduced into binary code as noises.

We innovatively choose global constant arrays as features targeting binary-to-source scene, including constant number arrays, constant enum arrays and constant string arrays. These constant data will be stored in `.data` or `.rdata` segment of binary code as a steam of binary data, that would not affected by compilation at all. At the same time, global constant arrays are always key constants in algorithms and core functions in general that reflact the uniqueness of a software.

We extracted this arrays from source code by parsing Abstract Syntax Tree (AST), recovering data structure of array and matrix, and finding out the specific value of each enum element and calculation statement (e.g., `int a[]={1*3, 2*3};`). Then we transfer these arrays to binary steam by traverse possible element length like 1/2/4/8 byte(s), and search binary steam in data segment of binary code.

2) Control Flow Features: Normal control flow features, such as control flow graph and function call graph, are commonly used in binary-to-binary comparison. But for binary code and source code, function structure would be greatly changed by function inlining and extra functions invoked by compiler such as stack protection. However, we still found some control flow features that are stable during compilation, namely complex switch/case and if/else structures.

Switch/case structures could be really complex and unique sometimes. Some of them have targeting branches that corresponds to multiple case entrances (e.g., `case 1: case 2: case 3: branch(); break;`), and some have a large amount of cases or branches that will generate a jump table in binary code. We parse complex switch/case structure from binary code and source code, correctly handle the default branch and change of case values caused by optimizer, store them in uniform expression and do exact matching.

The longest continuous-or-nested if/else comparison path of a function (e.g. `if(a>1) {if(a>10) {if(a>100) func();}}`) is a specific structural characteristic of control flow, that contains relatively complete information of comparisons, jumps and branches in a function. And the constant numbers and comparison operators of the longest path are stable during compilation while instructions are likely to have lots of changes. We find out the longest comparison path by traversing code blockes in topological order, parse constant number and comparison operator of each comparison, and genarated constant number list of the longest comparison path.

IV. A Large-Scale System

A. System Overview

Our system establishs the mapping relationship between binary software and open-source projects, and detects open-source license violations among them. There are five functional modules in our system as shown in Fig. 2, including PE file extractor, feature extractor, file attribute analyzer, code feature matcher and violation detector.

PE file extractor automatically decompress installation packages and interact with installers to obtain their executable running files. (Section IV-B) Feature extractor extracts file attribution, code features and project license information. File attribute analyzer builds and propagates library-to-attribute and attribute-to-attribute relationships to recognize which open-source project that a binary file belongs to. (Section III-B) Code feature matcher calculates the similarities of code features

of binary code and source code to identify the open-source components of a binary file. (Section III-C) Violation detector detects violations by looking up the reusage of open-source projects under GPL/AGPL only.

B. Automatical Executable Files Extractor

Software programs are always released as installation packages rather than executable files in general. In fact, installation package will release executable files to user's system based on system settings and user's configurations. This makes installing process portable and flexible, but impede obtaining executable files of binary software programs and analysing it.

Unpacking installation packages statically is an effective method of extraction. Some installer packers are commonly used and have corresponded unpackers, e.g., Inno Unpacker for Inno Setup. And some can be decompressed by 7-zip in brute force. However, most installation packages can not be unpacked straightly, as they are customized developed without universe unpacking tools or download executables from Internet during installing process. To parse PE files automatically, we try to install them by simulating user's interaction while installing.

Installing process is complex and diverse, but there are still some common patterns. For example, some need users to agree with their usage agreements, config various configurations, connect to the network and download files. Thus, it is possible to stimulate users interaction with installers. We identify text in buttons and send clicking message to specific buttons, such as "Agree" and "Next", to go to the next step. When the installer is processing or downloading files, we wait for it for seconds and keep monitoring the installing status by differentiating the instant screenshot with last window states. The status monitor can also judge whether last button click is effective, if not, it helps the stimulator to click next possible button.

V. Evaluation

In our empirical experiments, all binary software programs and open-source projects are crawled from a famous sofeware download site of China named Baidu Application Center [13], Github and Ubuntu Packages [14].

A. Project Name Labeling

Project name labeling is to label an executable file with the open-source project name based upon the file attributes. After extracting executable files on 9K installer packages (half of Baidu Application Center) and getting 230K PE files, we got a statistics of PE file attributes shown in Table IV. After regularization and deduplication, company name has an average repetition time of 16.25, while internal name's is 4.21, and the maximum repetition times of company name and internal name are 26385 and 185 respectively.

After analyzing the attributes with heuristic rules, we found total 372 pairs of library-to-attribute and attribute-to-attribute relationships, and successfully labeled 60K PE files with project name, accounted for 26% of the total PE file samples.

B. Code Feature Matching

We select two groups of binary software programs as our benchmark. The first group contains 23 released binary files of open-source projects. The second group are commonly-used closed-source software programs, including Foxit Reader and Audacity, whose components are already known. From the view of source code, we select 88 popular open-source projects that widely involve different functions, including network protocols, compression algorithms, image parsing, font parsing, etc., such as openssl, zlib, libpng, freetype and so on.

We compare each pair of binary software and open-source projects, doing 2200 comparisons totally. There should be 45 pairs of reuse relationships, and we have got 9 false positives and 2 false negatives at all. Combined with group 1 and 2, we have got an accuracy of 99.5% and a recall of 95.6% on average with a precision of 82.7%. We present a comparison of our system and BAT [7], the only existing tool which can map binary code to source code in Table V. Since the extension of code features and file attributes, our system has a remarkable higher recall than BAT without loss of precision.

TABLE IV
The Statistics of PE File Attributes of 230K PE Files. The average repetition time of each attribute is no less than 4.21 and the maximum repetition is no less than 185.

Attribute Name	Original Info	Regularized Info	Deduplicated Info	Average Repeat	Maximun Repeat
Internal Name	184K	139K	33K	4.21	185
Legal Copyright	160K	110K	12K	9.17	14308
Product Name	197K	150K	25K	6.00	6536
Company Name	185K	130K	8K	16.25	26385

TABLE V
Accuracy of Code Feature Matching. Compared with the only existing tool BAT [7] which can map binary code to source code, our recall is much higher.

	Our System			BAT [7]	
	Accuracy	Precision	Recall	Precision	Recall
Group 1	2016/2024 (99.6%)	22/27 (81.5%)	22/23 (95.6%)	82%	75%
Group 2	173/176 (98.3%)	21/25 (84.0%)	21/22 (95.4%)		
Average	2189/2200 (99.5%)	43/52 (82.7%)	43/45 (95.6%)		

C. Large-scale Binary-to-source Mapping

We deployed our system on 15 servers with 4-core Intel(R) Core(TM) i7-4712MQ CPU@2.30Hz, 4GB memory and 128GB drivers, and we collected 264 open-source projects as target projects from Github, Ubuntu Packages [14], and official websites of open-source projects, and crawled installation packages of 1000 software programs. It took 69 hours to do comparison, and 56.5 seconds for each pair of binary software and source project on average.

In this experience, we find 2270 pairs of reuse relationships related to 527 binary software programs. As shown in Table VI, the top 10 frequently used projects are zlib, libjpeg-turbo, flac, sqlite, tinyxml, qt, ffmpeg, libpng, openssl and libcurl.

We also evaluate the contribution of each feature by counting how many PE files are labeled or matched with open-source projects through it (Figure 3). It is interesting to find that the most common-used features, string and export, only accounted for 38.06% contribution in total, while other features, either file attributes or code features, contain equally important

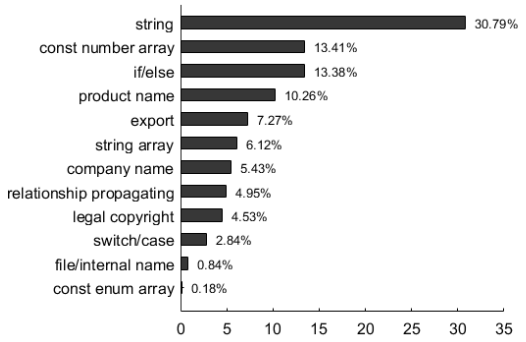

Fig. 3. Contribution of Each Feature. Commonly-used Features Only Contribute 38.06% While Our Innovative Features Contributes 61.94% in Total.

TABLE VI
THE TOP 10 OPEN-SOURCE PROJECTS IN REUSE FREQUENCY

Project Name	Reuse Frequency	Project Name	Reuse Frequency
zlib	384 (38.4%)	tinyxml	106 (10.6%)
libjpeg	257 (25.7%)	libpng	94 (9.4%)
sqlite	138 (13.8%)	ffmpeg	94 (9.4%)
openssl	124 (12.4%)	libtiff	84 (8.4%)
qt	112 (11.2%)	libjpeg	77 (7.7%)

information with traditional features while binary-to-source mapping.

D. License Violation Checking

Combined with binary-to-source mapping results and license information of open-source projects, it is capable to detect license violation of binary software. In this experience, we manually collect the license information of the top 100 frequently reused open-source projects. Because most common projects are under permissive licenses actually, and some projects under more restrictive copyleft licenses previously (e.g., FFmpeg and Qt) turn to support permissive licenses as well to compromise to secondary developers, we only find 9 open-source projects that only under GPL/AGPL right now, namely jbig2dec, game-music-emu, vsfilter, avisynth, rtmpdump, mplayer, lzo, mpeg2dec and x264. However, there are still 110 license violations of these 7 projects with an average of 10.3 violations per project (Table VII), involving 72 (7.2%) software programs.

VI. CONCLUSION

In this paper, we have presented a scalable system that can detect open-source license violation of binary software targeting GPL/AGPL. We take advantage of file attributes and code features that are less likely to be affected by compilation innovatively and systematically. We break the barrier of scalable analysis by implementing an executable file extractor. With an empirical experiment, we demonstrated out system has a high accuracy and recall without losing precision, and finally find 7.2% of 1000 binary software programs violated open-source licenses 110 times in total, related to 9 projects that are using GPL/AGPL license only. In the future work, we are planning to go deeper into the binary-to-source mapping problem and try to improve the precision of detection

TABLE VII
LICENSE VIOLATIONS OF EACH PROJECT

Project Name	Violations	Examples
mplayer	44	HUPlayer, Agogo Video to PSP Converter
x264	20	Easy CD-DA Extractor, 51Talk AC
jbig2dec	14	Advanced PDF Compressor, PDFrizator
mpeg2dec	12	Apex Video Converter
rtmpdump	11	GiliSoft Video Recorder
avisynth	5	Ultra Video Splitter
lzo	4	cb2bib

ACKNOWLEDGMENT

This research was supported by the National Science Foundation of China (NO. 61602470, NO. 61572481, NO. U1836209); the Program of Beijing Municipal Science & Technology Commission (NO. D181100000618004); the National Key Research and Development Program of China (2016QY071405); the Strategic Priority Research Program of the CAS (XDC02000000); the Program of Key Laboratory of Network Assessment Technology, the Chinese Academy of Sciences; and the Program of Beijing Key Laboratory of Network Security and Protection Technology.

REFERENCES

[1] *Open source license usage on GitHub.com*, https://blog.github.com/2015-03-09-open-source-license-usage-on-github-com. Last accessed 12 Aug 2018

[2] *Top 10 Open Source Software Licenses of 2016 and Key Trends*, https://resources.whitesourcesoftware.com/blog-whitesource/top-10-open-source-software-licenses-of-2016-and-key-trends. Last accessed 12 Aug 2018

[3] Mathur, A., Choudhary, H., Vashist, P., Thies, W., Thilagam, S. : *An Empirical Study of License Violations in Open Source Projects*. 2012 35th Annual IEEE Software Engineering Workshop, 168–176 (2012)

[4] Keivanloo, I., Forbes, C., Hmood, A., Erfani, M., Neal, C., Peristerakis, G., Rilling, J. : *A linked data platform for mining software repositories*. Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, 32–35 (2012)

[5] Wu, Y., Manabe, Y., Kanda, T., German, D., Inoue, K. : *A method to detect license inconsistencies in large-scale open source projects*. Proceedings of the 12th Working Conference on Mining Software Repositories, 324–333 (2015)

[6] Duan, R., Bijlani, A., Xu, M., Kim, T., Lee, W. : *Identifying Open-Source License Violation and 1-day Security Risk at Large Scale*. Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security, 2169–2185 (2017)

[7] Hemel, A., Kalleberg, K., Vermaas, R., Dolstra, E. : *Finding software license violations through binary code clone detection*. Proceedings of the 8th Working Conference on Mining Software Repositories, 63–72 (2011)

[8] Paschalides, D., Kapitsaki, G.M. : *Validate your SPDX files for open source license violations*. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 1047–1051 (2016)

[9] Kapitsaki, G.M., Kramer, F. : *Open Source License Violation Check for SPDX Files*. Software Reuse for Dynamic Systems in the Cloud and Beyond, 90–105 (2015)

[10] Saebjornsen, A., Willcock, J., Panas, T., Quinlan, D., Su, Z. : *Detecting code clones in binary executables*. Proceedings of the eighteenth international symposium on Software testing and analysis, 117–128 (2009)

[11] Pewny, J., Schuster, F., Bernhard, L., Holz, T. and Rossow, C. : *Leveraging semantic signatures for bug search in binary programs*. Proceedings of the 30th Annual Computer Security Applications Conference, 406–415 (2014)

[12] Eschweiler, S., Yakdan, K. and Gerhards-Padilla, E. : *discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code*. NDSS (2016)

[13] *Baidu Application Center*, http://rj.baidu.com. Last accessed 12 April 2018

[14] *Ubuntu - Ubuntu Package Search*, https://packages.ubuntu.com/. Last accessed 12 Aug 2018