SourceFinder: Finding Malware Source-Code from Publicly Available Repositories in GitHub

Md Omar Faruk Rokon
UC Riverside
mroko001@ucr.edu

Risul Islam
UC Riverside
risla002@ucr.edu

Ahmad Darki
UC Riverside
adark001@ucr.edu

Evangelos E. Papalexakis
UC Riverside
epapalex@cs.ucr.edu

Michalis Faloutsos
UC Riverside
michalis@cs.ucr.edu

Abstract

Where can we find malware source code? This question is motivated by a real need: there is a dearth of malware source code, which impedes various types of security research. Our work is driven by the following insight: public archives, like GitHub, have a surprising number of malware repositories. Capitalizing on this opportunity, we propose, SourceFinder, a supervised-learning approach to identify repositories of malware source code efficiently. We evaluate and apply our approach using 97K repositories from GitHub. First, we show that our approach identifies malware repositories with 89% precision and 86% recall using a labeled dataset. Second, we use SourceFinder to identify 7504 malware source code repositories, which arguably constitutes the largest malware source code database. Finally, we study the fundamental properties and trends of the malware repositories and their authors. The number of such repositories appears to be growing by an order of magnitude every 4 years, and 18 malware authors seem to be "professionals" with a well-established online reputation. We argue that our approach and our large repository of malware source code can be a catalyst for research studies, which are currently not possible.

1 Introduction

Security research could greatly benefit by an extensive database of malware source code, which is currently unavailable. This is the assertion that motivates this work. First, security researchers can use malware source code to: (a) understand malware behavior and techniques, and (b) evaluate security methods and tools. In the latter, having the source code can provide the groundtruth for assessing the effectiveness of different techniques, such as reverse engineering methods. Second, currently, a *malware source code* database is not readily available. By contrast, there are several databases with *malware binary code*, as collected via honeypots, but even those are often limited in number and not widely available. We discuss existing malware archives in Section 9.

A missed opportunity: Surprisingly, software archives, like GitHub, host many publicly-accessible malware repositories, but this has not yet been explored to provide security researchers with malware source code. In this work, we focus on GitHub which is arguably the largest software storing and sharing platform. As of October 2019, GitHub reports more than 34 million users [25] and more than 32 million public repositories [24]. As we will see later, there are thousands of repositories that have malware source code, which seem to have escaped the radar of the research community so far. We use a broad definition of malware to include any repository containing software that can participate in compromising devices and supporting offensive, undesirable and parasitic activities.

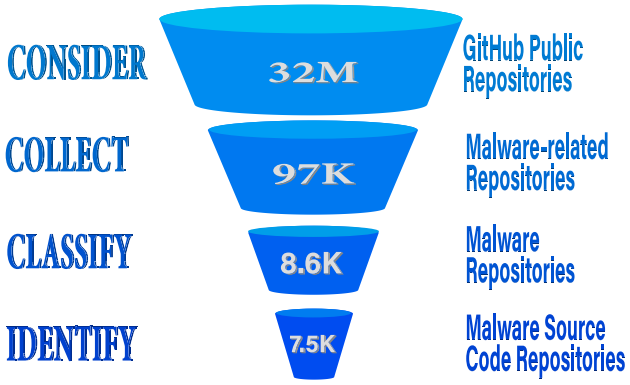


Figure 1: The steps of our work as a funnel: We identify 7.5K malware source code repositories in GitHub starting from 32M repositories based on 137 malware keywords (Q137).

Why do authors create public malware repositories? This question mystified us: these repositories expose both the creators and the intelligence behind the malware. Intrigued, we conducted a small investigation on malware authors, as we discuss below.

Problem: How can we find malware source code repositories in a large archive, like GitHub? The input to the problem is an online archive and the desired output is a database of

malware repositories. The challenges include: (a) collecting an appropriate set of repositories from the potentially vast archive, and (b) identifying the repositories that contain malware. Optionally, we also want to further help researchers that will potentially use these repositories, by determining additional properties, such as the most likely target platform, the malware type or family etc. Another practical challenge is the need to create the ground truth for validation purposes.

Related work: To the best of our knowledge, there does not seem to be any study focusing on the problem above. We group related works in the following categories. First, several studies analyze software repositories to find usage and limitations without any focus on malware [14]. Second, several efforts create and maintain databases of malware binaries but without source code [2, 3]. Third, many efforts attempt to extract higher-level information from binaries, such as lifting to Intermediate Representation (IR) [20], but it is really difficult to re-create the source code [10]. In fact, such studies would benefit from our malware source-code archive to evaluate and improve their methods. Taking a software engineering angle, an interesting work [8] compares the evolution of 150 malware source code repositories with that of benign software. We discuss related works in Section 9.

Contributions: Our work is arguably the first effort to systematically identify malware source code repositories from a massive public archive. The contribution of this work is threefold: (a) we propose SourceFinder, a systematic approach to identify malware source-code repositories with high precision, (b) we create, arguably, the largest non-commercial malware source code archive with 7504 repositories, and (c) we study patterns and trends of the repository ecosystem including temporal and author-centric properties and behaviors. We apply and evaluate our method on the GitHub archive, though it could also be used on other archives, as we discuss in Section 8.

Our key results can be summarized in the following points, and some key numbers are shown in Figure 1.

a. We collect **97K malware-related repositories** from GitHub, namely repositories retrieved using malware keywords through GitHub's API and employing techniques to overcome several limitations. We also generate an extensive groundtruth with 2013 repositories, as we explain in Section 3.

b. **SourceFinder achieves 89% precision.** We systematically consider different Machine Learning approaches, and carefully-created representations for the different fields of the repository, such as title, description etc. We then systematically evaluate the effect of the different features, as we discuss in Section 5. We show that we classify malware repositories with a 89% precision, 86% recall and 87% F1-score using five fields from the repository.

c. **We identify 7504 malware source-code repositories**, which is arguably the largest malware source-code database available to the research community. We have already downloaded the contents in these repositories, in case GitHub decides to deactivate them. We also create a curated database of 250 malware repositories, manually verified and spanning a wide range of malware types.

d. **The number of new malware repositories in our data more than triples every four years**. The increasing trend is interesting and alarming at the same time.

e. **We identify popular and influential repositories.** We study the malware repositories using three metrics of popularity: the number of watchers, forks and stars. We find 8 repositories that dominate the top-5 lists for all three metrics.

f. **We identify prolific and influential authors.** We find that 3% of the authors have more than 300 followers. We also find that 0.2% of the authors have more than 7 malware repositories, with the most prolific author *cyberthreats* having created 336 repositories.

g. **We identify and profile 18 professional hackers.** We find 18 authors of malware repositories, who seem to have created a brand around their activities, as they use the same user names in security forums. For example, user *3vilp4wn* (pronounced evil-pawn) is the author of a keylogger malware in GitHub, which the author is promoting in the *Hack This Site* forum using the same username. We present our study of malware authors in Section 7.

Open-sourcing for maximal impact: creating an engaged community. We intend to make our datasets and our tools available for research purposes at our website [28]. Our vision is to create community-driven reference platform, which will provide: (a) malware source code repositories, (b) community-vetted labels and feedback, and (c) open-source tools for collecting and analyzing malware repositories. Our goal is to expand our database with more software archives and richer information. Although authors could start hiding their repositories (see Section 8), we argue that our already-retrieved database could have significant impact in enabling certain types of security studies [22, 29, 32].

2 Background

We provide background information on GitHub and the type of information that repositories have.

GitHub is a massive world-wide software archive, which enables users to share code through its public repositories thus creating a global social network of interaction. For instance, first, users can collaborate on a repository. Second, users often "fork" projects: they copy and evolve projects. Third, users can follow projects, and "up-vote" projects using "stars" (think Facebook likes). Although GitHub has many private repositories, there are 32 million public software repositories.

We describe the key elements of a GitHub repository. A repository is equivalent to a project folder, and typically, each repository corresponds to a single software project. However, a repository could contain: (a) source code, (b) binary code, (c) data, (d) documents, such as latex files, and (e) all of the above.

A repository in GitHub has the following data fields: a) title,

b) description, c) topics, d) README file, e) file and folders, f) date of creation and last modified, g) forks, h) watchers, i) stars, and j) followers and followings, which we explain below.

a. Repository title: The title is a mandatory field and it usually consists of less than 3 words.

b. Repository description: This is an optional field that describes the objective of the project and it is usually 1-2 sentences long.

c. Repository topics: An author can optionally provide topics for her repository, in the form of tags, for example, *"linux, malware, malware-analysis, anti-virus"*. Note that 97% of the repositories in our dataset have less than 8 topics.

d. README file: As expected, the README file is a documentation and/or light manual for the repository. This field is optional and its size varies from one or two sentences to many paragraphs. For example, we found that 17.48% of the README files in our repositories are empty.

e. File and folders: In a well-constructed software, the file and folder names of the source code can provide useful information. For example, some malware repositories contain files or folders with indicative names, such as "malware", "source code" or even specific malware types or names of specific malware, like *mirai*.

f. Date of creation and last modification: GitHub maintains the date of creation and last modification of a repository. We find malware repository created in 2008 are actively being modified by authors till present.

g. Number of forks: Users can fork a public repository: they can create a clone of the project. An user can fork any public repository to change locally and contribute to the original project if the owner accepts the modification. The number of forks is an indication of the popularity and impact of a repository. Note that the number of forks indicates the number of distinct users that have forked a repository.

h. Number of watchers: Watching a repository is equivalent to "following" in the social media language. A "watcher" will get notifications, if there is any new activity in that project. The numbers of watchers is an indication of the popularity of a repository [16].

i. Number of stars: A user can "star" a repository, which is equivalent to the "like" function in social media [5], and places the repository in the users favorite group, but does not provide constant updates as with the "watching" function.

j. Followers: Users can also follow other users' work. If A follows B, A will be added to B's followers and B will be added to A's following list. The number of followers is an indication of the popularity of a user [39].

3 Data Collection

The first step in our work is to collect repositories from GitHub that have a higher chance of being related to malware. Extracting repositories at scale from GitHub hides several subtleties and challenges, which we discuss below.

Set	Descriptions	Size
Q1	Query set = {"malware"}	1
Q50	Query with 50 keywords with Q1⊂Q50	50
Q137	Query with 137 keywords with Q50⊂Q137	137
RD1	Retrieved repositories from query Q1	2775
RD50	Retrieved repositories from query Q50	14332
RD137	Retrieved repositories from query Q137	97375
LD1	Labeled subset of RD1 dataset	379
LD50	Labeled subset of RD50 dataset	755
LD137	Labeled subset of RD137 dataset	879
M1	Malware source code repositories in RD1	680
M50	Malware source code repositories in RD50	3096
M137	Malware source code repositories in RD137	7504
MCur	Manually verified malware source code dataset	250

Table 1: Datasets, their relationships, and their size.

Using the GitHub Search API, a user can query with a set of keywords and obtain the most relevant repositories. We describe briefly how we select appropriate keywords, retrieve related repositories from GitHub and how we establish our ground truth.

A. Selecting keywords for querying: In this step, we want to retrieve repositories from GitHub in a way that: (a) provides as many as possible malware repositories, and (b) provides a wide coverage over different types of malware. For this reason, we select keywords from three categories: (a) malware and security related keywords, such as malware and virus, (b) malware type names, such as ransomware and keylogger, and (c) popular malware names, such as mirai. Due to space limitations, we will provide the full list of keywords in our website at publication time for repeatability purposes.

We define three sets of keywords that we use to query GitHub. The reason is that we want to assess the sensitivity of the number of keywords on the outcome. Specifically, we use the following query sets: (a) the **Q1 set**, which only contains the keyword "malware"; (b) the **Q50 set**, which contains 50 keywords, and (c) the **Q137 set** which contains 137 keywords. The Q137 keyword set is a super-set of Q50, and Q50 is a superset of Q1. As we will see below, using the query set Q137 provides wider coverage, and we recommend in practice. We use the other two to assess the sensitivity of the results in the initial set of keywords. We list our datasets in Table 1.

B. Retrieving related repositories: Using the Search API, we query GitHub with our set of keywords. Specifically, we query GitHub with every keyword in our set separately. In an ideal world, this would have been enough to collect all related repositories: a query with "malware" (Q1) should return the many thousands related repositories, but this is not the case.

The search capability hides several subtleties and limitations. First, there is a limit of 1000 repositories that a single

Labeled Dataset	Malware Repo.	Benign Repo.
LD137	313	566
LD50	326	429
LD1	186	193

Table 2: Our groundtruth: labeled datasets for each of the three queries, for a total of 2013 repositories.

search can return: we get the top 1000 repositories ordered by relevancy to the query. Second, the GitHub API allows 30 requests per minute for an authenticated user and 10 requests per minute for an unauthenticated user.

Bypassing the API limitations. We were able to find a work around for the first limitation by using ranking option. Namely, a user can specify her preferred ranking order for the results based on: (a) best match, (b) most stars, (c) fewest stars, (d) most forks, (e) fewest forks, (f) most recently updated, and (g) the least recently updated order. By repeating a query with all these seven ranking options, we can maximize the number of distinct repositories that we get. This way, for each keyword in our set, we search with these seven different ranking preferences to obtain a list of GitHub repositories.

C. Collecting the repositories: We download all the repositories identified in our queries using PyGithub [52], and we obtain three sets of repositories RD1, RD50 and RD137. These retrieved datasets have the same "subset" relationship that they query sets have: RD1 $\subset$ RD50 $\subset$ RD137. Note that we remove pathological repositories, mainly repositories with no actual content, or repositories "deleted" by GitHub. For each repository, we collect and store: (a) repository-specific information, (b) author-specific information, and (c) all the code within the repository.

As we see from Table 1, using more and specialized malware keywords returns significantly more repositories. Namely, searching with the keyword "malware" does return 2775 repositories, but searching with the Q50 and Q137 returns 14332 and 97375 repositories respectively.

D. Establishing the groundtruth: As there was no available groundtruth, we needed to establish our own. As this is a fairly technical task, we opted for domain experts instead of Mechanical Turk users, as recommended by recent studies [23]. We use three computer scientists to manually label 1000 repositories, which we selected in a uniformly random fashion, from each of our dataset RD137 and RD50 and 600 repositories from RD1. The judges were instructed to independently investigate every repository thoroughly.

Ensuring the quality of the groundtruth. To increase the reliability of our groundtruth, we took the following measures. First, we asked judges to label a repository *only*, if they were certain that it is malicious or benign and distinct, and leave it unlabeled otherwise. We only kept the repositories for which the judges agreed unanimously. Second, duplicate repositories were removed via manual inspection, and were excluded from the final labeled dataset to avoid overfitting. It is worth noting that we only found very few duplicates in the order of 3-5 in each dataset with hundreds of repositories.

With this process, we establish three separate labeled datasets named LD137, LD50, and LD1 starting from the respective malware repositories from each of our queries, as shown in Table 2. Although the labeled datasets are not 50-50, they are representing both classes reasonably well, so that a naive solution that will label everything as one class, would perform poorly. By contrast, our approach performs sufficiently well, as we will see in Section 5.

As there is no available dataset, we argue that we make a sufficient size dataset by manual effort.

4 Overview of our Identification Approach

Here, we describe our supervised learning algorithm to identify the repositories that contain malware.

Step 1. Data preprocessing: As in any Natural Language Processing (NLP) method, we start with some initial processing of the text to improve the effectiveness of the solution. We briefly outline three levels of processing functionality.

a. Character level preprocessing: We handle the character level "noise" by removing special characters, such as punctuation and currency symbols, and fix Unicode and other encoding issues.

b. Word level preprocessing: We eliminate or aggregate words following the best practices of Natural Language Processing [33]. First, we remove article words and other words that don't carry significant meaning on their own. Second, we use a stemming technique to handle inflected words. Namely, we want to decrease the dimensionality of the data by grouping words with the same "root". For example, we group the words "organizing", "organized", "organize" and "organizes" to one word "organize". Third, we filter out common file and folder names that we do not expect to help in our classification, such as "LEGAL", "LICENSE", "gitattributes" etc.

c. Entity level filtering: We filter entities that are likely not helpful in describing the scope of a repository. Specifically, we remove numbers, URLs, and emails, which are often found in the text. We found that this filtering improved the classification performance. In the future, we could consider mining URLs and other information, such as names of people, companies or youtube channels, to identify authors, verify intention, and find more malware activities.

Step 2. The repository fields: We consider fields from the repositories that can be numbers or text. Text-based fields require processing in order to turn them into classification features and we explain this below. We use and evaluate the following text fields: title, description, topics, file and folder names and README file fields.

Text field representation: We consider two techniques to represent each text field by a feature in the classification.

a. Bag of Words (BoW): The bag-of-words (BoW) model is among the most widely used representations of a document. The document is represented as the number of occurrences of

its words, disregarding grammar and word order [75]. This model is commonly used in document classification where the frequency of each word is used as feature value for training a classifier [42]. We use the model with the count vectorizer and TF-IDF vectorizer to create the feature vector.

In more detail, we represent each text field in the repository with a vector $V[K]$, where $V[i]$ corresponds to the significance of word i for the text. There are several ways to assign values $V[i]$: (a) zero-one to account for presence, (b) number of occurrences, and (c) the TF-IDF value of the word. We evaluated all the above methods.

Fixing the number of words per field. To improve the effectiveness of our approach using BoW, we conduct a feature selection process, χ^2 statistic following best practices [55]. The χ^2 statistic measures the lack of independence between a word (feature) and a class. A feature with lower chi-square score is less informative for that class, and thus not useful in the classification. We discuss this further in Section 5. For each text-based field f, we select the top K_f words for that field, which exhibit the highest discerning power in identifying malware repositories. Note that we set a value for K_f during the training stage For each field, we select the value K_f, as we explain in Section 5.

b. Word embedding: The word embedding model is a vector representations of each word in a document: each word is mapped to an M-dimensional vector of real numbers [44], or equivalently are projected in an M-dimensional space. A good embedding ensures that words that are close in meaning have nearby representations in the embedded space. In order to create the document vector, word embedding follows two approaches (i) frequency-based vectorizer(unsupervised) [58] and (ii) content-based vectorizer(supervised) [38]. Note that in this type of representation, we do not use the *word level processing*, which we described in the previous step, since this method can leverage contextual information.

We use frequency-based word embedding with word average and TF-IDF vectorizer. We also use pre-trained model of Google word2vec [43] and Stanford (Glov) [49] to create the feature vector.

Finally, we create the vector of the repository by concatenating the vectors of each field of that repository.

Step 3. Selecting the fields: Another key question is which fields from the repository to use in our classification. We experiment with all of the fields listed in Section 2 and we explain our findings in the next Section.

Step 4. Selecting a ML engine: We design ML model to classify the repositories into two classes: (i) malware repository and (ii) benign repository. We systematically evaluate many machine learning algorithms [7,45]: Naive Bayes (NB), Logistic Regression (LR), Decision Tree (CART), Random Forest(RF), K-Nearest Neighbor (KNN), Linear Discriminant Analysis (LDA), and Support Vector Machine (SVM).

Step 5. Detecting source code repositories: In this final step, we want to identify the presence of source code in the repositories. By June 2020, GitHub started labeling repositories that contain source code. Therefore, one can simply filter out all repositories that are not labelled as such.

As our study predates this GitHub feature, we developed a heuristic approach to identify source code repositories independently, which we describe below. Our heuristic exhibits 100% precision as validated by GitHub's classification, as we will see in Section 5.

Our source-code classification heuristics works in two steps. First, we identify files in the repository that contain source code. To do this, we start by examining their file extension. If the file extension is one of the known programming languages: *Assembly, C, C++, Batch File, Bash Shell Script, Power Shell Script, Java, Python, C#, Objective-C, Pascal, Visual Basic, Matlab, PHP, Javascript, and Go*, we label it as a source file. Second, if the number of source files in a repository exceeds the **Source Percentage threshold (SourceThresh)**, we consider that the repository contains source code.

5 Evaluation: Choices and Results

In this section, we evaluate the effectiveness of the classification based on the proposed methodology defined in Section 4. More specifically, our goal here is to answer the following questions:

1. **Repository field selection:** Which repository fields should we consider in our analysis?
2. **Field representation:** Which feature representation is better between bag of words (BoW) and word embedding and considering several versions of each?
3. **Feature selection**: What are the most informative features in identifying malware repositories?
4. **ML algorithm selection:** Which ML algorithm exhibits the best performance?
5. **Classification effectiveness:** What is the precision, recall and F1-score of the classification?
6. **Identifying malware repositories:** How many malware repositories do we find?
7. **Identifying malware source code repository:** How many of the malware repositories have source code?

Note that we have a fairly complex task: we want to identify the best fields, representation method and Machine Learning engine, while considering different values for parameters. What complicates matters is that all these selections are interdependent. We present our analysis in sequence, but we followed many trial and error and non-linear paths in reality.

1. Selecting repository fields: We evaluated all the repository fields mentioned earlier. In fact, we used a significant number of experiments with different subsets of the features, not shown here due to space limitations. We find that the title, description, topics, README file, and file and folder names have the most discerning power. We also considered number of forks, watchers, and stars of the repository and the number

Representation	Classification Accuracy Range
Bag of Words with Count Vectorizer	86%-51%
Bag of Words with Count Vectorizer + Feature Selection	91%-56%
Bag of Words with TF-IDF vectorizer	82%-63%
Word Embedding with Word Average	85%-72%
Word Embedding with TF-IDF	85%-74%
Pretrained Google word2vec Model	76%-64%
Pretrained Stanford (Glov) Model	73%-62%

Table 3: Selecting the feature representation model: We evaluate all the representations across seven machine learning approaches and report the range of the overall accuracy.

of followers and followings of the author of the repository. We found that not only it did not help, but it usually decreased the classification accuracy by 2-3%. One possible explanation is that the numbers of forks, stars and followers reflect the popularity rather than the content of a repository.

2. Selecting a field representation: The goal is to find, which representation approach works better. In Table 3, we show the comparison of the range of classification accuracy across the 7 different ML algorithms that we will also consider below. We find that Bag of Words with the count vectorizer representation reaches 86% classification accuracy, with the word embedding approach nearly matching that with 85% accuracy. Note that we finetune the selection of words to represent each field in the next step.

Why does not the embedding approach outperform the bag of words? One would have expected that the most complex embedding approach would have been the winner and by a significant margin. We attribute this to the relatively small text size in most text fields, which also do not provide well-structured sentences (think two-three words for the title, and isolated words for the topics). Furthermore, the word co-occurrences does not exist in the topics and file names fields, which is partly what makes embedding approaches work well in large and well structured documents [26, 41].

In the rest of this paper, we use the Bag of Words with count vectorizer to represent our text fields, since it exhibits good performance and is computationally less intensive than the embedding method.

3. Fixing the number of words per field. We want to identify the most discerning words from each text field, which is a standard process in NLP for improving the scalability, efficiency and accuracy of a text classifier [12]. Using the χ^2 statistic, we select the top K_f best words from each field.

To select the appropriate number of words per field, we followed the process below. We vary K_f = 5,10,20,30,40 and 50 for title, topic and README file, and we find that the top 30 words in title, 10 words in topic and 10 words in README file exhibit the highest accuracy. Similarly, we try K_f = 80, 90, 100, 110 and 120 for file names and K_f = 300, 325, 350, 375, 400, 425, 450 and 475 for the description field. We find that the top 100 words for file and folder names and top 400 words for description field give the highest accuracy. Note that we do this during training and refining the algorithm, and then we continue to use these words as features in testing.

Thus, we select the top: (a) 30 words from the title, (b) 10 words from the topics, (c) 400 words from the description, (d) 100 words from the file names, and 10 words from the README file. This leads to a total of 550 words across all fields. For reference, we find 9253 unique words in the repository fields of our training dataset. Reducing the focus on the top 550 most discerning words per field increases the classification accuracy by as much as 20% in some cases.

4. Evaluating and selecting ML algorithms: We find that Multinomial Naive Bayes exhibits the best F1-score with 87%, striking a good balance between 89% precision and 86% recall for the malware class among other machine learning classifier which we considered. Detecting the benign class, we do even better with 92% precision, 94% recall and 93% F1-score. By contrast, the F1-score of the other algorithms is below 79%. Note that KNN, LR and LDA methods provide higher precision, but with significantly lower recall. Thus, one could use these algorithms to get higher precision at the cost of lower total number of repositories.

We use Multinomial Naive Bayes as our classification engine for the rest of this study. We attempt to explain the superior F1-Score of the Naive Bayes in our context. The main advantage of Naive Bayes over other algorithms is that it considers the features independently of each other for a given class and can handle large number of features better. As a result, it is more robust to noisy or unreliable features. It also performs well in domains with many equally important features, where other approaches suffer, especially with a small training data, and it is not prone to overfitting [64]. As a result, the Naive Bayes is considered a dependable algorithm for text classification and it is often used as the benchmark to beat [71].

5. Assessing the effect of the query set: We have made the following choices in the previous steps: (a) 5 text-based fields, (b) bag of words with count vectorization, (c) 550 total words across all the fields, and (d) the Multinomial Naive Bayes. We perform 10-fold cross validation and report the precision, recall and F1-score in Figure 2 for our three different labeled data sets. We see that the precision stays above 89% for all three datasets, with a recall above 77%.

It is worth noting the relative stability of our approach with respect to the keyword set for the initial query especially between LD50 and LD137 datasets. The LD1 dataset we observe higher accuracy, but significantly less recall compared to LD137. We attribute this fact to the single keyword used in selecting the repositories in LD1, which may have lead to a more homogeneous group of repositories. Interestingly, LD50 seem to have the lower recall and F1-score even though the differences are not that large.

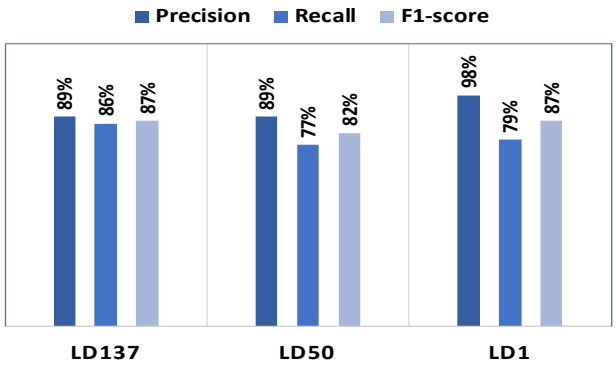


Figure 2: Assessing the effect of the number of keywords in the query: Precision, Recall and F1-score of our approach on the LD137, LD50 and LD1 labeled datasets.

Dataset	Initial	Malware	Mal. + Source
RD1	2775	809	680
RD50	14332	3615	3096
RD137	97375	8644	7504

Table 4: The identified repositories per dataset with: (a) malware, and (b) malware and source code.

6. Identifying 8644 malware repositories: We use LD137 to train our Multinomial Naive Bayes model and apply it on RD137 dataset. We find 8644 malware repositories. We also apply the same trained model on RD1 and RD50 and find 809 and 3615 malware repositories respectively, but this repositories are included in the 8644. (Recall that RD1 and RD50 are subsets of RD137).

7. Identifying 7504 malware source code repositories: As of June 2020, we can use the source code labelling to identify such repositories. Here, we use this labelling to validate our heuristic approach for completeness.

In deploying our heuristic, we set our Source Percentage threshold to 75%, meaning that: if more than 75% of files in a repository are source code files, we label it as a source code repository. Applying this heuristic, we find that 7504 repositories are most likely source code repositories in RD137. We use the name **M137** to refer to this group of malware source code repositories. We find 680 and 3096 malware source code repositories in RD1 and RD50 as shown in Table 4. However, these are subset of M137, given that RD1 and RD50 are subsets of RD137.

We find that 100% of our source code repositories are also labeled as such by GitHub. We argue that our heuristic could be useful for other software archives, which may not provide the "source code" label.

8. A curated malware source code dataset: MCur As a tangible contribution, we provide, MCur, a dataset of 250 repositories from the M137 dataset, which we manually verify for containing malware source code and relating to a particular malware type. Opting for diversity and coverage, the dataset spans all the identified types: virus, backdoor, botnet, keylogger, worm, ransomware, rootkit, trojan, spyware, spoof, ddos, sniff, spam, and cryptominer. We intend to constantly update and make our labeled malware repositories publicly available [28].

6 A large scale study of malware

Encouraged by the substantial number of malware repositories, we study the distributions and longitudinal properties of the identified malware repositories in M137.

Caveat: We provide some key observations in this section, but they should be viewed as indicative and approximate trends and only within the context of the collected repositories and with the general assumption that repository titles and descriptions are reasonably accurate. In Section 8, we discuss issues around the biases and limitations that our dataset may introduce.

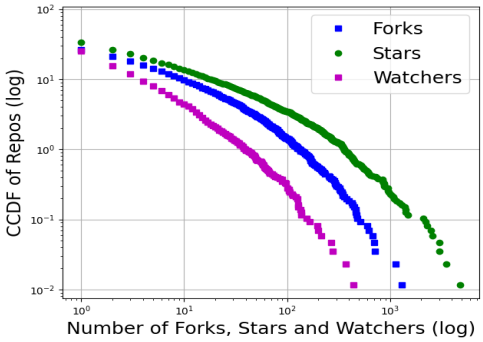


Figure 3: CCDF distributions of forks, stars and watchers per repository.

A. Identifying influential repositories. The prominence of a repository can be measured by the number of *forks*, *stars*, and *watchers*. In Figure 3, we plot the complementary cumulative distribution function (CCDF) of these three metrics for our malware repositories.

Fork distribution: We find that 2% of the repositories seem quite influential with at least 100 forks as shown in Figure 3. Recall that the fork counter indicates the number of distinct users that have forked a repository. For reference, 78% of the repositories have less than 2 forks.

Star distribution: We find that 2% of the repositories receive more than 250 stars as shown in Figure 3. For reference, 75% of the repositories have less than 3 stars.

Watcher distribution: In Figure 3, we find that 1% of the repositories have more than 50 watchers. For reference, we observe that 84% of the repositories have less than 3 watchers. Note that these distributions are skewed, and follow patterns that can be approximated by a log-normal distribution.

Which are the most influential repositories? We find that 8 repositories dominate the top 5 spots across all three metrics:

R ID	Author	# Star	# Fork	# Watcher	Content of the Repository
1	ytisf	4851	1393	730	80 malware source code and 140 Binaries
2	n1nj4sec	4811	1307	440	Pupy RAT
3	Screetsec	3010	1135	380	TheFatRat Backdoor
4	malwaredllc	2515	513	268	Byob botnet
5	RoganDawes	2515	513	268	USB attack platform
6	Visgean	626	599	127	Zeus trojan horse
7	Ramadhan	535	283	22	30 malware samples
8	dana-at-cp	1320	513	125	backdoor-apk backdoor

Table 5: The profile of the top 5 most influential malware repositories across all three metrics with 8 unique repositories.

stars, forks, and watchers. We present a short profile of these dominant repositories in Table 5. Most of the repositories contain a single malware project, which is an established practice among the authors in GitHub [48,66]. We find that the repository "theZoo" [72], created by *ytisf* in 2014 is the most forked, watched, and starred repository with 1393 forks, 730 watchers and 4851 stars as of October, 2019. However, this repository is quite unique and was created with the intention of being a malware database with 140 binaries and 80 source code repositories.

Influence metrics are correlated: As one would expect, the influence and popularity metrics are correlated. We use a common correlation metric, the Pearson Correlation Coefficient (r) [6], measured in a scale of $[-1,1]$. We calculate the metric for all pairs of our three popularity metrics. We find that all of them exhibit higher positive correlation: stars vs. forks ($r = 0.92$, $p < 0.01$), forks vs. watchers ($r = 0.91$, $p < 0.01$) and watchers vs. stars ($r = 0.91$, $p < 0.01$).

B. Malware type and target platform. We wanted to get a feel for what type of malware we have identified. As a first approximation, we use the keywords found in the text fields to relate repositories in M137 with the type of malware and the intended target platform. Our goal is to create the two-dimensional distribution per malware type and the target platform as shown in Table 6. To create this table, we associate a repository with keywords in its title, topics, descriptions, file names and README file fields of: (a) the 6 target platforms, and (b) the 13 malware type keywords.

How well does this heuristic approach work? We provide two different indications of its relative effectiveness. First, the vast majority of the repositories relate to one platform or type of malware: (a) less than 8% relate to more than one platform, and (b) less than 11% relate to more than one type of malware. Second, we manually verify the 250 repositories in our curated data MCur and find a 98% accuracy.

Below, we provide some observations from Table 6.

a. Keyloggers reign supreme. We see that one of the largest categories is the keylogger malware with 679 repositories, which are mostly affiliated with Windows and Linux platforms. We discuss the emergence of keyloggers below in our temporal analysis.

Types	Target Platform						
	Wind.	Linux	Mac	IoT	Andr.	iOS	**Total**
Total	**1592**	**1365**	**380**	**108**	**442**	**131**	**4018**
keylogger	396	209	42	2	27	3	**679**
backdoor	181	227	37	11	51	4	**511**
virus	235	131	34	2	51	16	**469**
botnet	153	154	43	36	64	17	**467**
trojan	133	70	24	16	67	19	**329**
spoof	76	115	88	2	20	9	**310**
rootkit	55	163	13	2	19	3	**255**
ransomware	117	67	14	1	33	13	**245**
ddos	71	95	20	10	9	3	**208**
worm	61	45	18	5	25	18	**172**
spyware	45	22	6	6	38	16	**133**
spam	40	29	18	14	23	5	**129**
sniff	29	38	23	1	15	5	**111**

Table 6: Distribution of the malware repositories from M137 dataset based on the malware type and malware target platform. This table demonstrates the repositories that fit with the criteria defined in Section 6.

b. Windows and Linux are the most popular targets. Not surprisingly, we find that the majority of the malware repositories are affiliated with these two platforms: 1592 repositories for Windows, and 1365 for Linux.

c. MacOS-focused repositories: fewer, but they exist. Although MacOS platforms are less commonly targeted, we find a non-trivial number of malware repositories for MacOS. As shown in Figure 4c, there are 380 MacOS malware repositories, which is roughly an order of magnitude less compared to those for Windows and Linux.

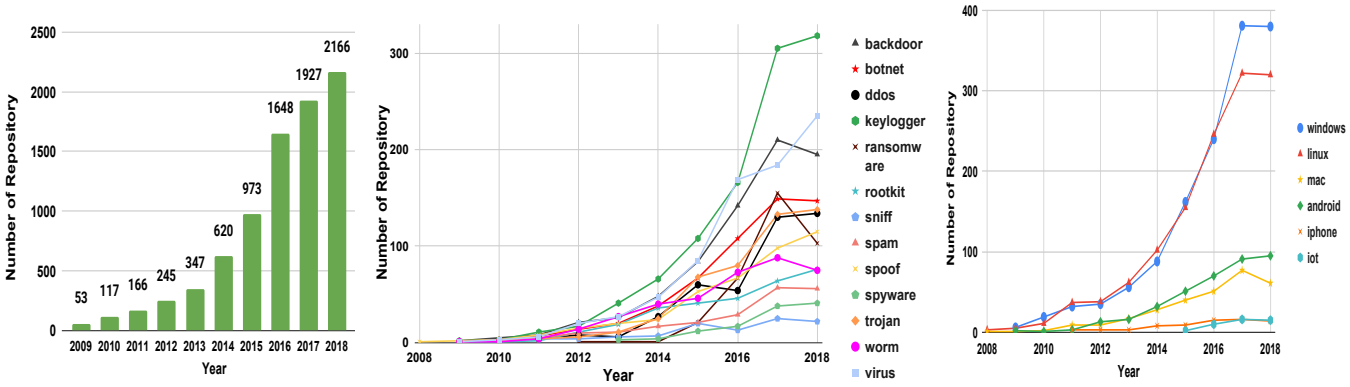
C. Temporal analysis. We want to study the evolution and the trends of malware repositories. We plot the number of new malware repositories per year: a) total malware, b) per type of malware, and c) per target platform in Figure 4. We discuss a few interesting temporal behaviors below.

a. The number of new malware repositories more than triples every four years. We see an alarming increase from 117 malware repositories in 2010 to 620 repositories in 2014 and to 2166 repositories in 2018. We also observe a sharp increase of 70% between 2015 to 2016 shown in Figure 4a.

b. Keyloggers started a super-linear growth since 2010 and are by far affiliated with the most new repositories per year since 2013, but their rate of growth reduced in 2018.

c. Ransomware repositories emerge in 2014 and gain momentum in 2017. Ransomware experienced their highest growth rate in 2017 with 155 new repositories, while that number dropped to 103 in 2018.

d. Malware activity slowed down in 2018 across the board. It seems that 2018 is a slower year for all malware even when seen by type (Figure 4b) and target platform (Fig-



(a) New malware repositories created per year.

(b) New repositories per type of malware per year.

(c) New malware repositories per target platform per year.

Figure 4: New malware repositories per year: a) all malware, b) per type of malware, and c) per target platform.

ure 4c). We find that the number of new malware repositories has dropped significantly in 2018 for most types of malware except virus, keylogger and trojan.

e. IoT and iPhone malware repositories become more visible after 2014. We find that IoT malware emerges in 2015 and iPhone malware sees an increase after 2014 in Figure 4c. We conjecture that this is possibly encouraged by the emergence and increasing popularity of specific malware: (a) WireLurker, Masque, AppBuyer malware [13] for iPhones, and (b) BASHLITE [70], a Linux based botnet for IoT devices. We find the names of the aforemntioned malware in many repositories starting in 2014. Interestingly, the source code of the original BASHLITE botnet is available in a repository created by *anthonygtellez* in 2015.

f. Windows and Linux: dominant but slowing down. In Figure 4c, we see that windows and linux malware are flattened between 2017 and 2018. By contrast, IoT and android repositories have increased.

7 Understanding malware authors

Intrigued by the fact that authors create public malware repositories, we attempt to understand and profile their behavior.

As a first step towards understanding the malware authors, we want to assess their popularity and influence. We use the following metrics: (a) number of malware repositories which they created, (b) number of followers, (c) total number of watchers on their repositories, and (d) total number of stars. We focus on the first two metrics here. We use the notation *top k authors* for any of the metrics above, where *k* can be any positive integer to referring to "heavy-hitters".

A. Finding influential malware authors. We study the distribution of the number of malware repositories created and the number followers per author in following.

First, we find that 15 authors are contributing roughly 5% of all malware repositories by examining the CCDF of the created repositories in Figure 5. From the figure, we find an

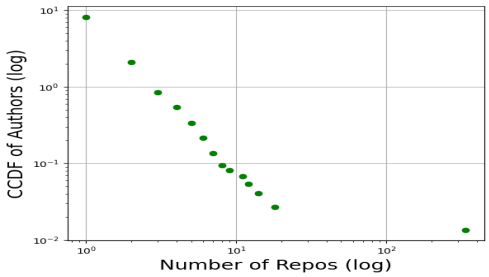


Figure 5: CCDF of malware repositories per author.

outlier author, *cyberthreats*, who doesn't follow power law distribution [21], has created 336 malware repositories. We also find that 99% authors have less than 5 repositories.

Second, we study the distribution of the number of followers per author, but omit the plot due to space limitations. The distributions is skewed with 3% (221) of the authors having more than 300 followers each, while 70% of the authors have less than 16 followers.

B. Malware authors strive for an online "brand": In an effort to understand the motive of sharing malware repositories, we make the following investigation.

a. Usernames seem persistent across online platforms. We find that many malware authors use the same username consistently across many online platforms, such as security forums. We conjecture that they are developing a reputation and they use their username as a "unique" identifier.

We identify 18 malware authors[1], who are active in at least one of the three security forums: Offensive Community, Ethical Hacker and Hack This Site, for which we happen to have access to their data. We conjecture that at least some of these usernames correspond to the same users based on the follow-

[1]Note that this does not mean that the other authors are not doing the same, but they could be active in other security forums or online platforms.

ing two indications. First, we find direct connections between the usernames across different platforms. For example, user *3vilp4wn* at the "Hack This Site" forum is promoting a keylogger malware by referring to a GitHub repository [1] whose author has the same username. Second, these usernames are fairly uncommon, which increases the likelihood of belonging to the same person. For example, there is a GitHub user with the name *fahimmagsi*, and someone with the same username is boasting about their hacking successes in the "Ethical Hacker" forum. As we will see below, *fahimmagsi* seems to have a well-established online reputation.

b. "Googling" usernames reveals significant hacking activities. Given that these GitHub usernames are fairly unique, it was natural to look them up on the web at large. Even a simple Internet search with the usernames reveals significant hacking activities, including hacking websites or social networks, and offering hacking tutorials in YouTube.

We investigate the *top 40* most prolific malware authors using a web search with a *single* simple query: "hacked by <username>". We then examine only the first page of search results. Despite all these self-imposed restrictions, we identify three users with substantial hacking related activities across Internet. For example, we find a number of news articles for hacking a series of websites by GitHub users *fahimmagsi* and *CR4SH* [65] [15]. Moreover, we find user *n1nj4sec* sharing a multi-functional Remote Access Trojan (RAT) named "Pupy", developed by her, which received significant news coverage in security articles back in March of 2019 [46] [54]. We are confident that well-crafted and targeted searches can connect more malware authors with hacking activities and usernames in other online forums.

8 Discussion

We discuss the effectiveness and limitations of SourceFinder.

a. Why is malware publicly available in the first place? Our investigation in Section 7 provides strong indications that malware authors want to actively establish their hacking reputation. It seems that they want to boost their online credibility, which often translates to money. Recent works [18, 51, 57] study the underground markets of malware services and tools: it stands to reason that notorious hackers will attract more clients. At the same time, GitHub acts as a collaboration platform, which can help hackers improve their tools.

b. Do we identify every malware repository in GitHub? Our tool can not guarantee that it will identify every malware repository in GitHub. First, we can only identify repositories that "want to be found": (a) they must be public, and (b) they must be described with the appropriate text and keywords. Clearly, if the author wants to hide her repository, we won't be able to find it. However, we argue that this defeats the purpose of having a public archive: if secrecy was desired, the code would have been shared through private links and services. Second, our approach is constrained by GitHub querying limitations, which we discussed in Section 3, and the set of 137 keywords that we use. However, we are encouraged by the number and the reasonable diversity of the retrieved repositories we see in Table 6.

c. Are our datasets representative? This is the typical hard question for any measurement or data collection study. First of all, we want to clarify that our goal is to create a large database of malware source code. So, in that regard, we claim that we accomplished our mission. At the same time, we seem to have a fair number of malware samples in each category of interest, as we see in Table 6.

Studying the trends of malware is a distant second goal, which we present with the appropriate caveat. On the one hand, we are limited by GitHub's API operation, as we discussed earlier. On the other hand, we attempt to reduce the biases that are under our control. To ensure some diversity among our malware, we added as many words as we could in our 137 malware, which is likely to capture a wide range of malware types. We argue that the fairly wide breadth of malware types in Table 6 is a good indication. Note that our curated dataset MCur with 250 malware is reasonably representative in terms of coverage.

d. What is the overlap among the identified repositories? Note that our repository does not include forked repositories, since **GitHub does not return forked repositories as answers to a query**. Similarly, the breadth of the types of the malware as shown in Table 6 hints at a reasonable diversity. However, our tool cannot claim that the identified repositories are distinct nor is it attempting do so. GitHub does not restrict authors from copying (downloading), and uploading it as a new repository. In the future, we intend to study the similarity and evolution among these repositories.

e. Are the authors of repositories the original creator of the source code? This is an interesting and complex question that goes beyond the scope of this work. Identifying the original creator will require studying the source code of all related repositories, and analyzing the dynamics of the hacker authors, which we intend to do in the future.

f. Are all the malware authors malicious? Not necessarily. This is an interesting question, but it is not central to the main point of our work. On the one hand, we find some white hackers or researchers, such as Yuval Nativ [74], or Nicolas Verdier [47]. On the other hand, several authors seem to be malicious, as we saw in Section 7.

g. Are our malware repositories in "working order"? It is hard to know for sure, but we attempt to answer indirectly. First, we pick 30 malware source codes and all of them compiled and a subset of 15 of them actually run successfully in an emulated environment as we already mentioned. Second, these public repositories are a showcase for the skills of the author, who will be reluctant to have repositories of low quality. Third, public repositories, especially popular ones, are inevitably scrutinized by their followers.

h. Can we handle evasion efforts? Our goal is to create the largest malware source-code database possible and having

collected 7504 malware repositories seems like a great start. In the future, malware authors could obfuscate their repositories by using misleading titles, and description, and even filenames. We argue that authors seem to want their repositories to be found, which is why they are public. We also have to be clear: it is easy for the authors to hide their repositories, and they could start by making them private or avoid GitHub altogether. However, both these moves will diminish the visibility and "street-cred" of the authors.

i. Will our approach generalize to other archives? We believe that SourceFinder can generalize to other archives, which provide public repositories, like GitLab and BitBucket. We find that these sites allow public repositories and let the users retrieve repositories. We have also seen equivalent data fields (title, description, etc). Therefore, we are confident that our approach can work with other archives.

9 Related Work

There are several works that attempt to determine if a piece of software is malware, usually focusing on a binary, using static or dynamic analysis [4, 17, 36, 60]. However, to the best of our knowledge, no previous study has focused on identifying malware source code in public software archives, such as GitHub, in a systematic manner as we do in this work. We highlight the related works in the following categories:

a. Studies that need malware source code. Several studies [40, 62, 78] use malware source code that are manually retrieved from GitHub repositories. Some studies [8] [9] compare the evolution and the code reuse of 150 malware source codes (with only some from GitHub) with that of benign software from a software engineering perspective and study the code reuse. Overall, various studies [22, 32] can benefit from malware source code to fine-tune their approach.

b. Mining and analyzing GitHub: Many studies have analyzed different aspects of GitHub, but not with the intention of retrieving malware repositories. First, there are efforts that study the user interactions and collaborations on GitHub and their relationship to other social media in [30, 37, 50]. Second, some efforts discuss the challenges in extracting and analyzing data from GitHub with respect to sampling biases [14, 27]. Other works [34, 35] study how users utilize the various features and functions of GitHub. Several studies [31, 53, 67] discuss the challenges of mining software archives, like *SourceForge* and GitHub, arguing that more information is required to make assertions about users and software projects. Finally, some efforts [61, 63, 76, 77] study GitHub repositories, but they focus on establishing a systematic method for identifying similarities, and use it to identify classes of repositories (e.g. Android versus web applications). Most of these studies use topic modeling, which is one of the approaches that we considered initially, but gave poor results in our context, but we will revisit in the future.

c. Databases of malware source code: At the time of writing this paper, there are few malware source code databases and are rarely updated such as project *theZoo* [72]. To the best of our knowledge, there does not exist an active archive of malware source code, where malware research community can get an enough number of source code to analyze.

d. Databases of malware binaries: There are well established malware binary collection initiatives, such as Virustotal [68] which provides analysis result for a malware binary. There are also community based projects such as VirusBay [69] that serve as malware binary sharing platform.

e. Converting binaries to source code: A complementary approach is to try to generate the source code from the binary, but this is a very hard task. Some works [19, 20] focus on reverse engineering of the malware binary to a high-level language representation, but not source code. Some other efforts [11, 29, 59] introduce binary decompilation into readable source code. However, malware authors use sophisticated obfuscation techniques [56] [10, 73] to make it difficult to reverse engineer a binary into source code.

f. Measuring and modeling hacking activity. Some other studies analyze the underground black market of hacking activities but their starting point is security forums [18, 51, 57], and as such they study the dynamics of that community but without retrieving any malware code.

10 Conclusion

Our work capitalizes on a great missed opportunity: there are thousands of malware source code repositories on GitHub. At the same time, there is a scarcity of malware source code, which is necessary for certain research studies.

Our work is arguably the first to develop a systematic approach to extract malware source-code repositories at scale from GitHub. Our work provides two main tangible outcomes: (a) we develop SourceFinder, which identifies malware repositories with 89% precision, and (b) we create, possibly, the largest non-commercial malware source code archive with 7504 repositories. Our large scale study provide some interesting trends for both the malware repositories and the dynamics of the malware authors.

We intend to open-source both SourceFinder and the database of malware source code to maximize the impact of our work. Our ambitious vision is to become the authoritative source for malware source code for the research community by providing tools, databases, and benchmarks.

Acknowledgements

This work was supported by the UC Multicampus National Lab Collaborative Research and Training (UC NL CRT) award #LFR18548554.

References

[1] 3vilp4wn. Hacking tool of 3vilp4wn. https://github.com/3vilp4wn/CryptLog/. [Online; accessed 08-February-2020].

[2] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 468–471. IEEE, 2016.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[4] John Aycock. *Computer viruses and malware*, volume 22. Springer Science & Business Media, 2006.

[5] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Software*, 30(1):52–66, 2013.

[6] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[8] Alejandro Calleja, Juan Tapiador, and Juan Caballero. A look into 30 years of malware development from a software metrics perspective. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 325–345. Springer, 2016.

[9] Alejandro Calleja, Juan Tapiador, and Juan Caballero. The malsource dataset: Quantifying complexity and code reuse in malware development. *IEEE Transactions on Information Forensics and Security*, 14(12):3175–3190, 2018.

[10] Gengbiao Chen, Zhengwei Qi, Shiqiu Huang, Kangqi Ni, Yudi Zheng, Walter Binder, and Haibing Guan. A refined decompiler to generate c code with high readability. *Software: Practice and Experience*, 43(11):1337–1358, 2013.

[11] Gengbiao Chen, Zhuo Wang, Ruoyu Zhang, Kan Zhou, Shiqiu Huang, Kangqi Ni, and Zhengwei Qi. A novel lightweight virtual machine based decompiler to generate c/c++ code with high readability. *School of Software, Shanghai Jiao Tong University, Shanghai, China*, 11, 2010.

[12] Jingnian Chen, Houkuan Huang, Shengfeng Tian, and Youli Qu. Feature selection for text classification with naïve bayes. *Expert Systems with Applications*, 36(3):5432–5435, 2009.

[13] Chris Stobing. ios malwares in 2014. https://www.digitaltrends.com/computing/decrypt-2014-biggest-year-malware-yet/. [Online; accessed 08-February-2020].

[14] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Findings from github: methods, datasets and limitations. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 137–141. IEEE, 2016.

[15] CR4SH. Hacking tool of cr4sh. https://github.com/Cr4sh/s6_pcie_microblaze/. [Online; accessed 08-February-2020].

[16] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286. ACM, 2012.

[17] Anusha Damodaran, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H Austin, and Mark Stamp. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, 13(1):1–12, 2017.

[18] Ashok Deb, Kristina Lerman, Emilio Ferrara, Ashok Deb, Kristina Lerman, and Emilio Ferrara. Predicting Cyber-Events by Leveraging Hacker Sentiment. *Information*, 9(11):280, nov 2018.

[19] Lukás Ďurfina, Jakub Křoustek, and Petr Zemek. Psybot malware: A step-by-step decompilation case study. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 449–456. IEEE, 2013.

[20] Lukáš Ďurfina, Jakub Křoustek, Petr Zemek, Dušan Kolář, Tomáš Hruška, Karel Masařík, and Alexander Meduna. Design of a retargetable decompiler for a static platform-independent malware analysis. In *International Conference on Information Security and Assurance*, pages 72–86. Springer, 2011.

[21] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review*, 29(4):251–262, 1999.

[22] Sri Shaila G, Ahmad Darki, Michalis Faloutsos, Nael Abu-Ghazaleh, and Manu Sridharan. Idapro for iot malware analysis? In *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*, Santa Clara, CA, August 2019. USENIX Association.

[23] Joobin Gharibshah, Evangelos E Papalexakis, and Michalis Faloutsos. Rest: A thread embedding approach for identifying and classifying user-specified information in security forums. *arXiv preprint arXiv:2001.02660*, 2020.

[24] GitHub. Repository search for public repositories: Showing 32,107,794 available repository results. https://github.com/search?q=is:public/. [Online; accessed 13-October-2019].

[25] GitHub. User search: Showing 34,149,146 available users. https://github.com/search?q=type:user&type=Users/. [Online; accessed 13-October-2019].

[26] Amir Globerson, Gal Chechik, Fernando Pereira, and Naftali Tishby. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, 8(Oct):2265–2295, 2007.

[27] Georgios Gousios and Diomidis Spinellis. Mining software engineering data from github. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 501–502. IEEE, 2017.

[28] HackerChatter. UCR hacker forum webtool for extracting useful information from security forums! http://www.hackerchatter.org/. [Online; accessed 22-July-2020].

[29] Richard Healey. Source code extraction via monitoring processing of obfuscated byte code, August 27 2019. US Patent 10,394,554.

[30] Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O Hall, and Adriana Iamnitchi. Mentions of security vulnerabilities on reddit, twitter and github. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 200–207. ACM, 2019.

[31] James Howison and Kevin Crowston. The perils and pitfalls of mining sourceforge. In *MSR*, pages 7–11. IET, 2004.

[32] James A Jerkins. Motivating a market or regulatory solution to iot insecurity with the mirai botnet code. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–5. IEEE, 2017.

[33] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[34] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101. ACM, 2014.

[35] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035–2071, 2016.

[36] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *USENIX security symposium*, volume 4, pages 351–366, 2009.

[37] Bence Kollanyi. Automation, algorithms, and politics: Where do bots come from? an analysis of bot codes shared on github. *International Journal of Communication*, 10:20, 2016.

[38] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966, 2015.

[39] Michael J Lee, Bruce Ferwerda, Junghong Choi, Jungpil Hahn, Jae Yun Moon, and Jinwoo Kim. Github developers use rockstars to overcome overflow of news. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 133–138. ACM, 2013.

[40] Toomas Lepik, Kaie Maennel, Margus Ernits, and Olaf Maennel. Art and automation of teaching malware reverse engineering. In *International Conference on Learning and Collaboration Technologies*, pages 461–472. Springer, 2018.

[41] Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[42] Michael Frederick McTear, Zoraida Callejas, and David Griol. *The conversational interface*, volume 6. Springer, 2016.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[44] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[45] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[46] n1nj4sec. Pupy tool. https://github.com/n1nj4sec/pupy/wiki/. [Online; accessed 08-February-2020].

[47] Nicolas Verdier. Security researcher. https://www.linkedin.com/in/nicolas-verdier-b23950b6/. [Online; accessed 14-February-2020].

[48] Nikhil Gupta. Should we create a separate git repository of each project or should we keep multiple projects in a single git repo? https://www.quora.com/. [Online; accessed 14-February-2020].

[49] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[50] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: sentiment analysis of security discussions on github. In *Proceedings of the 11th working conference on mining software repositories*, pages 348–351. ACM, 2014.

[51] Rebecca S. Portnoff, Sadia Afroz, Greg Durrett, Jonathan K. Kummerfeld, Taylor Berg-Kirkpatrick, Damon McCoy, Kirill Levchenko, and Vern Paxson. Tools for Automated Analysis of Cybercriminal Markets. *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, pages 657–666, 2017.

[52] PyGithub. A python libraray to use github api v3. https://github.com/PyGithub/PyGithub/. [Online; accessed 13-October-2019].

[53] Austen Rainer and Stephen Gale. Evaluating the quality and quantity of data on open source software projects. In *Procs 1st int conf on open source software*, 2005.

[54] Raj Chandel. Article on pupy. https://www.hackingarticles.in/command-control-tool-pupy/. [Online; accessed 08-February-2020].

[55] Monica Rogati and Yiming Yang. High-performing feature selection for text classification. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 659–661, 2002.

[56] Hassen Saıdi, Phillip Porras, and Vinod Yegneswaran. Experiences in malware binary deobfuscation. *Virus Bulletin*, 2010.

[57] Anna Sapienza, Sindhu Kiranmai Ernala, Alessandro Bessi, Kristina Lerman, and Emilio Ferrara. Discover: Mining online chatter for emerging cyber threats. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, pages 983–990, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee.

[58] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 298–307, 2015.

[59] Eric Schulte, Jason Ruchti, Matt Noonan, David Ciarletta, and Alexey Loginov. Evolving exact decompilation. In *Workshop on Binary Analysis Research (BAR)*, 2018.

[60] Madhu K Shankarapani, Subbu Ramamoorthy, Ram S Movva, and Srinivas Mukkamala. Malware detection using assembly and api call sequences. *Journal in computer virology*, 7(2):107–119, 2011.

[61] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 314–319, 2017.

[62] Victor RL Shen, Chin-Shan Wei, and Tony Tong-Ying Juang. Javascript malware detection using a high-level fuzzy petri net. In *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, volume 2, pages 511–514. IEEE, 2018.

[63] Marcus Soll and Malte Vosgerau. Classifyhub: an algorithm to classify github repositories. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 373–379. Springer, 2017.

[64] SL Ting, WH Ip, and Albert HC Tsang. Is naive bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5(3):37–46, 2011.

[65] Tom K. Hacking news of fahim magsi. https://www.namepros.com/threads/hacked-by-muslim-hackers.950924/. [Online; accessed 08-February-2020].

[66] Tommy Hodgins. Choosing between "one project per repository" vs "multiple projects per repository" architecture. https://hashnode.com/. [Online; accessed 14-February-2020].

[67] Christoph Treude, Larissa Leite, and Maurício Aniche. Unusual events in github repositories. *Journal of Systems and Software*, 142:237–247, 2018.

[68] Virus Total. Free online virus, malware and url scanner. https://www.virustotal.com/en. [Online; accessed 08-February-2020].

[69] VirusBay. A web-based, collaboration platform for malware researcher. https://beta.virusbay.io/. [Online; accessed 08-February-2020].

[70] Wikipedia. Linux based botnet bashlite. https://en.wikipedia.org/wiki/BASHLITE/. [Online; accessed 08-February-2020].

[71] Shuo Xu. Bayesian naïve bayes classifiers to text classification. *Journal of Information Science*, 44(1):48–59, 2018.

[72] Y. Nativ and S. Shalev. thezoo: A live malware repository. https://github.com/ytisf/theZoo. [Online; accessed 08-February-2020].

[73] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 158–177. IEEE, 2016.

[74] Yuval Nativ. Security researcher. https://morirt.com/. [Online; accessed 14-February-2020].

[75] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[76] Yu Zhang, Frank F Xu, Sha Li, Yu Meng, Xuan Wang, Qi Li, and Jiawei Han. Higitclass: Keyword-driven hierarchical classification of github repositories. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 876–885. IEEE, 2019.

[77] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23. IEEE, 2017.

[78] Xingsi Zhong, Yu Fu, Lu Yu, Richard Brooks, and G Kumar Venayagamoorthy. Stealthy malware traffic-not as innocent as it looks. In *2015 10th International Conference on Malicious and Unwanted Software*, pages 110–116. IEEE, 2015.