Journal of Computer Virology and Hacking Techniques
https://doi.org/10.1007/s11416-024-00512-6

ORIGINAL PAPER

“Dirclustering”: a semantic clustering approach to optimize website structure discovery during penetration testing

Diego Antonelli[1] · Roberta Cascella[1] · Antonio Schiano[1] · Gaetano Perrone[2] · Simon Pietro Romano[2]

Received: 12 April 2023 / Accepted: 30 December 2023

Abstract
Dirbusting is a technique used to brute force directories and file names on web servers while monitoring HTTP responses in order to enumerate server contents. Such a technique uses lists of common words to discover the hidden structure of the target website. Dirbusting typically relies on response codes as discovery conditions to find new pages. It is widely used in web application penetration testing, an activity that allows companies to detect website vulnerabilities. Dirbusting techniques are both time and resource-consuming, and innovative approaches have never been explored in this field. We hence propose “Dirclustering”, an advanced technique to optimize the dirbusting process by leveraging semantic clustering. Specifically, we use semantic clustering techniques to organize wordlist items in different groups according to their semantic meaning. The created clusters are used in an ad-hoc implemented next-word intelligent strategy. This paper demonstrates that clustering techniques outperform the commonly used brute-force methods. Performance is evaluated by testing eight different web applications. Results show a performance increase that is up to 50% for each of the conducted experiments.

Keywords Artificial intelligence · Dirbusting · Network security · Penetration testing · Performance assessment · Semantic clustering

1 Introduction

Web Application Penetration Testing (WAPT) is a proactive strategy employed by companies to identify vulnerabilities in web applications using a black-box approach. The process encompasses various phases, including information gathering, enumeration, exploitation, and analysis. Within the information gathering phase, exploring the web application’s structure and identifying libraries and frameworks are crucial components. OWASP [1] provides a standard methodology for testing web application security, emphasizing the significance of discovering the application’s structure in the “Fingerprinting Web Application” Security Test (WSTG-INFO-09).

Despite the black-box nature of Web Application Penetration Testing, where testers lack a priori knowledge about the web application’s structure, techniques such as spidering and dirbusting are employed. Spidering automates the process of analyzing internal links within HTML pages. However, the discovery of hidden links calls for alternative approaches. Dirbusting is a technique that involves brute-forcing a target with predictable folder and file names. It proves valuable in discovering hidden pages by monitoring HTTP responses. In this context, the choice of wordlists in dirbusting becomes critical, determined by factors like developer conventions, framework or CMS (Content Management System) used, and the web application’s development language.

Although the technique is used extensively by security experts, it requires time-consuming manual efforts to select proper wordlists. At the current time of writing, no research

✉ Gaetano Perrone
gaetano.perrone@unina.it

Diego Antonelli
diego.antonelli@nttdata.com

Roberta Cascella
roberta.cascella@nttdata.com

Antonio Schiano
antonio.schiano@nttdata.com

Simon Pietro Romano
spromano@unina.it

1 NTT DATA ITALIA S.P.A., VIA ERNESTO CALINDRI, MILANO, (MI), Italy

2 Department of Electrical Engineering and Information Technology (DIETI), University of Naples Federico II, Naples, Italy

works have attempted to analyze if it is possible to automatically select such wordlists in order to optimize the performance of the dirbusting process. This study aims to fill the gap by introducing "dirclustering", a novel approach to dirbusting optimization through semantic clustering. The paper demonstrates the effectiveness of this strategy in emulating the decision-making process of security experts when selecting wordlists.

We conduct a preliminary experiment based on eight well-known web applications that shows a performance improvement close to 50% compared to legacy brute-force approaches. The remainder of this paper is structured as follows. Section 2 delves into the details of the dirbusting process and provides an introduction to semantic clustering approaches. Section 3 shows related works using semantic clustering techniques for cybersecurity purposes, while in Section 4 our approach is illustrated. The designed experimental campaign is described, along with related results, in Section 5 and Section 6, respectively. The last two sections summarize the results and provide details about future evolutions of the proposed work.

2 Setting the context

In this section, we introduce the main concepts behind the dirbusting process, providing information about the techniques it employs. Additionally, we discuss semantic clustering along with the most well-known approaches it leverages.

2.1 Dirbusting

Dirbusting is a technique used to brute force a target with predictable folder and file names while monitoring HTTP responses to enumerate server contents. This technique uses wordlists to send HTTP requests to a target website and discover hidden pages. It is useful during the first phase of a Penetration Testing activity to discover the target application's structure. It is important to remark that Penetration Testing is usually carried out as a black-box activity. The security expert has no access to information about the web application under test and typically has just low-privileged access to the system. For this reason, she/he cannot see all the pages of a web application by just using spidering. Thus, one of the goals of dirbusting is to discover pages that are not visible by using standard spidering techniques. Hidden pages might allow a security expert to find sensitive content on the website or valid entry points to perform other vulnerability injection tests. Dirbusting accepts a properly constructed list of words as input and starts sending HTTP requests to the website to discover new pages. In order to successfully complete its task, the dirbusting process needs a proper discovery condition. A common approach is to use the response code for that purpose: a new page is found when an HTTP response contains a status code other than 404 (Page Not Found). In this work, we define "valid requests" as those HTTP requests that have a response code other than 404. The choice of wordlists plays a crucial role in obtaining good results in terms of discovered pages. Such a choice depends on the acquired knowledge about the web application. Security experts choose the wordlists based on several criteria, such as:

- which convention the developer has used to define paths;
- which framework/CMS (Content Management System) has been used;
- which language has been adopted to develop the web application.

If the web application contains a page whose name comes with camel case notation (e.g., *loginPage*), it is advisable to use camel case wordlists (*logoutPage*, *adminPage*, etc.). Similarly, if the fingerprinting phase detected the existence of a Wordpress Content Management System, an optimized wordlist should contain Wordpress-specific words (*wp-login.php*, *wp-logout.php*, etc.). On the other hand, if the web application contains files with well-known extensions (e.g., JSP, PHP), it is better to use a wordlist whose stems properly fit them.

In this work, we demonstrate that a *semantic clustering* strategy is able to optimize dirbusting activities by correctly mimicking the behavior of a security expert when it comes to choosing the most appropriate wordlist. Before delving into the details of the proposed approach, we will briefly introduce semantic clustering in the following subsection.

2.2 Semantic clustering

Clustering is the process of partitioning a set of data objects into subsets in such a way that items in the same group are more similar to each other than to those in other groups. The objective is to maximize intra-cluster similarity while at the same time minimizing inter-cluster similarity. It is a widely used technique in data mining for text domains, where the items to be clustered are textual, and they can be of different granularity (documents, paragraphs, sentences, or terms).

Simple text clustering algorithms represent textual information as a document-term matrix. Features are computed based on term frequencies, and semantically related terms are not considered. Thus, documents clustered in this way are not conceptually similar to one another if no terms are shared, as semantic relationships are ignored.

Semantic clustering, instead, consists in grouping items into semantically related groups [2, 3]. This requires measuring the semantic similarity between textual information, which can be accomplished by vectorizing the text corpus using, among the others, one of the following resources:

- Semantic networks like WordNet [4]: a large lexical database of more than 200 languages. Nouns, verbs, adjectives, and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked using conceptual-semantic and lexical relations.
- Word embedding techniques such as Word2Vec [5, 6] and GloVe [7]: a word embedding is a simple neural network trained to reconstruct the linguistic contexts of words. Its input is a large corpus of words and produces a vector space, with each unique word assigned to a corresponding vector. Word embeddings allow us to use an efficient, dense representation in which similar words have a similar encoding.
- Sentence Embeddings: while word embeddings encode words into a vector representation, sentence embeddings represent a whole sentence in a way that a machine can easily work with. These are capable of encoding a whole sentence as one vector. Examples are Doc2Vec [8], an adaptation of word2vec for documents, or more recent approaches such as the Universal Sentence Encoder (USE) [9] and InferSent [10].
- Language representation models like the Bidirectional Encoder Representations from Transformers (BERT) [11]: BERT is a method for pre-training language representations, meaning that a general-purpose "language understanding" model is trained on a large text corpus (e.g., Wikipedia), and then used for downstream Natural Language Processing tasks. Pre-trained representations can either be context-free or contextual. Context-free models such as word2vec [5, 6] or GloVe [7] generate a single word embedding representation for each word in the vocabulary, so, for example, the word *basket* would have the same representation in *sports* and *e-commerce*. Contextual models, instead, generate a representation of each word that depends on the other words in the sentence.

Our approach leverages the Universal Sentence Encoder (USE) [9] as the chosen sentence embedding technique. Indeed, one of the main tasks for training a USE encoder is identifying the semantic textual similarity (STS) [12] between sentence pairs scored by Pearson correlation with human judgments. This is a task that perfectly fits our needs.

After the text corpus encoding phase, it is required to use a clustering algorithm in order to create the semantic clusters. Several clustering techniques can be effective for this purpose [13], and among the available choices, the K-means algorithm is used for its simplicity and accuracy.

One of the issues with K-means is the effective choice of the parameter K, i.e., the number of target clusters. In our case, such an issue is solved by leveraging the well-known *elbow method*, a heuristic used in determining the number of clusters in a data set. Figure 2 provides a graphical representation of such a heuristic.

This paper demonstrates that the proposed method allows us to improve dirbusting techniques by leveraging artificial intelligence. The approach was tested on eight web applications with 30 repetitions each, demonstrating a substantial performance improvement in each case..

3 Related works

The use of artificial intelligence techniques to improve security tasks is well proven, especially for realizing anomaly detection [14–18] and malware detection systems [19–22]. Despite the extensive literature, to the best of our knowledge, the exploration of how to leverage artificial intelligence in dirbusting has not been undertaken before. Relevant works typically fall in the wider area of semantic clustering methodologies that have been extensively explored in other application domains [2, 3]. Regardless of their application to dirbusting, Natural Language Processing techniques have been extensively used in security. Karbab [23] uses Natural Language Processing and machine learning techniques to create a behavioral data-driven malware detection tool. Malhotra [24] shows that NLP can help evaluate the completeness, contradiction, and inconsistency of security requirements of a software system.

In general, the use of Artificial Intelligence techniques for Penetration Testing has not been fully explored yet. However, several techniques, such as fuzzing, have been used in other domains. As an example, in the software testing field, several works show how it is possible to optimize fuzzing techniques by using machine learning [25]. Our work shows how it is possible to optimize a bruteforce technique (i.e., dirbusting) by using Artificial Intelligence. Hitaj [26] demonstrates that a Deep Learning approach can outperform both rule-based and state-of-the-art password guessing methods. Since password guessing is fundamentally a brute-force attack, our work asserts that it is feasible to enhance Penetration Testing tasks through AI. Specifically, Natural Language Processing techniques have the potential to improve tasks related to word usage. With special reference to dirbusting, semantic clustering can indeed optimize a brute-force approach by finding both syntactic and semantic relations among words.

Semantic clusters can be modeled in different ways, including the usage of external resources such as Wikipedia like in [27, 28], where authors clustered the text corpus with an ensemble approach using knowledge and concepts from Wikipedia. Other works [29–32] leverage semantic net-

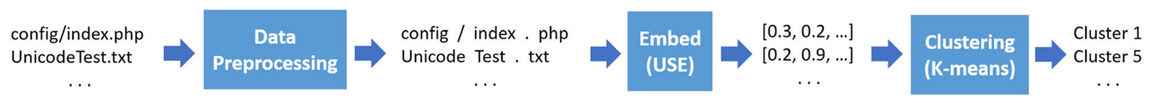


Fig. 1 The semantic clustering process used to group similar words. Each word in the wordlist is encoded by using the Universal Sentence Encoding (USE), then K-means clustering is used to group similar words

works, such as WordNet [4], which is used as word sense disambiguation to capture the main theme of the text and identify relationships among words. More recent applications use word embedding techniques [33–35] and sentence embedding techniques [36–38] where unsupervised embeddings models are used to encode the text corpus prior to the clustering process.

Our semantic clustering approach follows the works described in [36] and [37]. In [36], Universal Sentence Encoding (USE) [9] and InferSent [10] are used to find semantic similarities among user questions and cluster them by using the K-means clustering algorithm. In [37], such techniques are instead used to group similar tweets in a semantic sense.

4 Proposed solution

The proposed approach groups common files and directories contained in the wordlist based on the chosen semantic clustering technique. Semantic clusters define the execution order of the entries in the wordlist, with the aim of optimizing the dirbusting process. Namely, the approach involves a data pre-processing step on the entries of the wordlist and the subsequent creation of semantic clusters using the Universal Sentence Encoder (USE), in conjunction with the K-means clustering algorithm, as shown in Fig. 1.

4.1 Semantic clustering

The first step of the clustering process is data-processing, which consists of splitting each entry of the wordlist according to the naming convention (*camelCase, snake_case, kabab-case*), and punctuation characters: `( ! " $ # % & ` ( ) * +, -. /:; <= >? [ ] ^% _ { ∼ @)`.

For instance, the sentence *"comments/add_comment.php"* becomes *"comments / add _ comment. php"* and *"UnicodeTest.txt"* becomes *"Unicode Test. txt"*. This is a fundamental step because the naming convention and the punctuation characterizing each entry of the wordlist affect the encoding of the entry itself into embedding vectors. This may lead to a wrong similarity measure. For this reason, we detect the words in each entry to treat them as sentences instead of single words. This approach allows us to capture the semantic similarity among names contained in a wordlist, irrespective of the specific naming convention adopted by developers.

Then, a sentence embedding technique is used to encode each wordlist entry as a 512-dimensional vector so that similar words, often used in similar contexts, have a similar embedding vector representation.

More specifically, the *Universal Sentence Encoder (USE)* [9], version 4, implemented in TensorFlow 2.2.0 [39] is used. This model encodes text into high-dimensional vectors used for text classification, semantic similarity, clustering, and other natural language tasks. The model is trained on a variety of data sources and optimized for greater-than-word length text, such as sentences, phrases, or short paragraphs. One of the main tasks for the USE training is identifying the semantic textual similarity (STS) between sentence pairs, a task that perfectly fits our needs and justifies our choice.

Finally, extracted embeddings are used with clustering techniques to create the semantic clusters. Our approach uses the K-means clustering technique for its simplicity and accuracy. The number of clusters is chosen using the elbow method, a heuristic used to determine the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters and picking a point slightly right to the elbow of the curve as the number of clusters to use, 20 in this case.

In Fig. 3, Principal Component Analysis (PCA) is used to show in a two-dimensional space the similarities of the words of the wordlist encoded using USE. Each of the points in the picture represents a word (*word_ik*), and each color represents the cluster (*cluster_k*) where the word belongs. As shown in the picture, semantically similar words are closer in the embedding space and are grouped in the same cluster.

Other examples are presented in Table 1 where words belonging to 5 different clusters are analyzed.

4.2 Intelligent dirbusting strategy

We implemented an intelligent dirbusting strategy that uses the semantic clusters created according to the proposed approach to improve the legacy brute force techniques.

Commonly used dirbusting techniques require a huge number of requests to the target website, attempting to guess the names or identifiers of hidden functionality based on the common files and directories contained in the wordlists.

The choice of the wordlist depends on the information gathered by the security experts during the spidering process, whose main task is to enumerate the target's visible content and functionality. Based on the knowledge acquired during

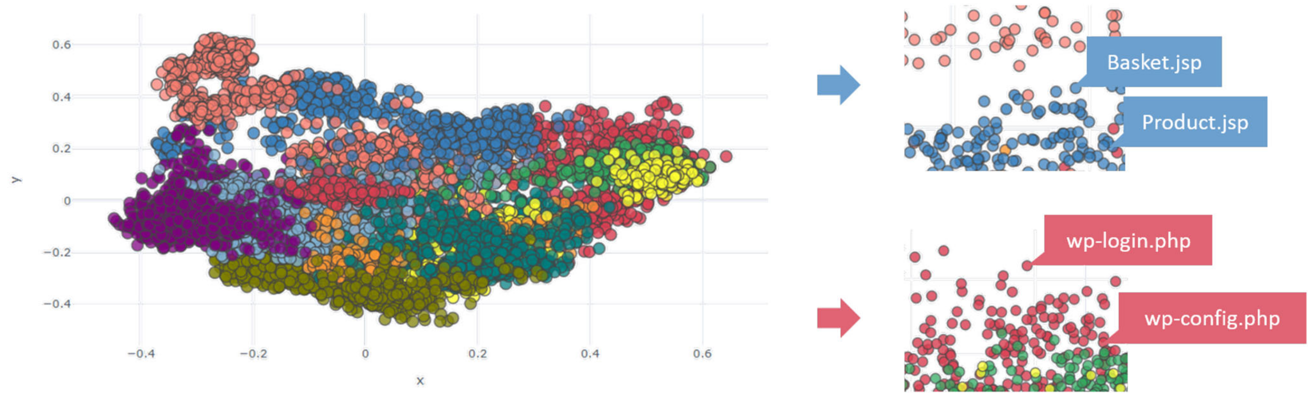


Fig. 3 The figure shows semantic clusters in a two-dimensional space generated by the Principal Component Analysis (PCA). The space shows the similarities of the words in the wordlist encoded using the Universal Sentence Encoder (USE)

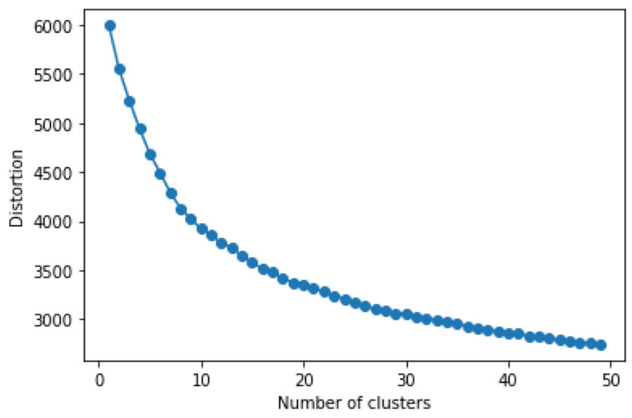


Fig. 2 Elbow method, used to select the optimal number of clusters. The number of clusters corresponding to the "elbow" point is considered to be optimal. It is a compromise between the number of clusters and the quality of clustering

this phase, the experts select the proper wordlist following these criteria:

- *Naming conventions*: developers are used to following a naming convention (camel case, snake case, kebab case) when implementing a web application. For this reason, if a security expert identifies a specific naming convention, the wordlist is chosen or adapted to it.
- *Content Management Systems (CMS)*: if the fingerprinting phase detected the existence of a certain CMS, a corresponding wordlist is chosen.
- *Used programming language*: if the web application under test contains files with well-known extensions (e.g. *.jsp*, *.php*), it is advisable to use a wordlist whose stems properly fit them.

As a general rule, in order to discover as much hidden content as possible, it is fundamental to choose the wordlist that best suits the target's characteristics. Once the wordlist is chosen, the dirbusting process starts by addressing HTTP requests to the target according to the directory and file names in the list. The order of execution of the requests follows the wordlist order, as described in algorithm 1 below.

Our approach aims at making dirbusting more intelligent by leveraging artificial intelligence. As described in the flow in Fig. 4, the process starts by choosing a random word (*word_ik*) from a common wordlist. When a valid URL is detected, the proposed strategy consists of choosing the cluster (*cluster_k*) where the current word belongs. In this way, the next words picked from the chosen cluster will likely tar-

Table 1 Examples of similarities in semantic clustered words

Word	Cluster
Libraries/joomla/github/package/gitignore.php	1
Plugins/user/joomla/joomla.php	1
Libraries/cms/menu/menu.php	1
Libraries/cms/component/helper.php	1
Wp-login.php	2
Wp-config.php	2
Wp-includes/fonts/dashicons.eot	2
Wp-content/plugins/index.php	2
About.php	3
Appinfo.php	3
Index.php	3
Update.php	3
Basket.jsp	4
Product.jsp	4
Search.jsp	4
Cart/.gitignore	4
Images/bricks.jpg	5
Images/menu/menu_tabs.gif	5
Misc/tree.png	5
Favicon.ico	5

```
while words in wordlist do
    pop a word from wordlist (word_i);
    url = basePath + word_i;
    statusCode = httpRequest(url);
    if statusCode is not 404 then
        save word_i;
    else
        continue iterations;
    end
end
```
Algorithm 1: Dirbusting Process. Send an HTTP request and check the response code for each word in the wordlist. If the response code is different from 404 (i.e., the "not found" status code), then store the path.

get another valid URL. Examples of words grouped in the same cluster are given in Tab. 1.

While common dirbusting techniques require that the expert is forced to manually select a wordlist according to the target characteristics, the intelligent dirbusting strategy accomplishes this task by building the above-described semantic clusters while considering the following aspects:

- the CMS used in web applications: as shown in Table 1, in clusters 1 and 2, words related to different CMSs are grouped together. In cluster 1, Joomla-related words are included, while in cluster 2 we can find words commonly used in WordPress;
- web application programming languages: clusters 3 and 4, in Table 1, include words related to specific languages. In these clusters, the programming languages *PHP* and *JSP*, respectively, are represented;
- semantic similarities: semantic clusters are able to consider semantic similarities as well. In this way, dirbusting is able to automatically understand the context of the website. In Tab. 1, cluster 4 contains words related to *e-commerce*, whereas in cluster 5, words associated with *images* are considered.

5 Trials and experimentation

The experiments aim to demonstrate that a dirbusting strategy based on semantic clustering enhances the discovery of a website structure by reducing the number of HTTP requests required to successfully complete the entire process. To this purpose, we developed a virtualized environment composed of 8 distinct target websites. Then, we created a wordlist containing full paths of each website and instrumented a dirbusting tool that can be configured to run either in bruteforce mode or by leveraging the semantic clustering strategy introduced by us. In order to create the wordlist, we have started all of the target applications and retrieved full paths by exe-

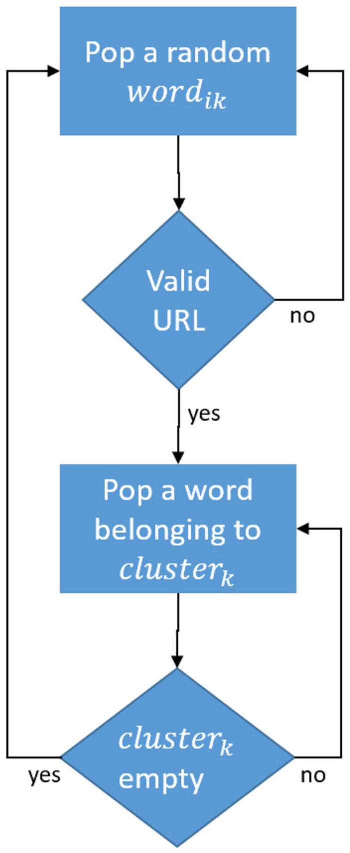


Fig. 4 The semantic clustering strategy flow is used to optimize the dirbusting process. At the start, a random word is chosen from a common wordlist. Then, depending on the found valid URLs, the word selection extracts words belonging to the cluster obtained by the semantic clustering process

Table 2 Web applications words count

Web Application	Words Count	Total %
Bodgeit	40	0.47%
Bricks	66	0.78%
Drupal (CMS)	1074	12.82%
DVWS	80	0.95%
Joomla (CMS)	4672	55.78%
Wacko	126	1.50%
Wordpress (CMS)	1595	19.04%
XVWA	722	8.62%
Total	8375	100%

cuting OS-level commands. In Tab. 2, the word count for each web application under attack is reported.

After obtaining the wordlist, we applied semantic clustering in order to group words according to their semantic meaning. The resulting clusters are stored in an ad hoc configuration file that is used by the instrumented dirbusting module when carrying out the clustering-based testing campaigns.

Finally, for each website, we have performed the experiments by executing the dirbusting tool both in *bruteforce* mode and in *semantic clustering* mode. Each experiment's results have been logged to enable further offline analysis of the collected data.

The virtualization environment is a useful alternative to a real-world setup for several reasons:

- we do not have to deal with network issues that might affect the environment;
- we do not create potential Denial Of Service conditions. Indeed, as the dirbusting process sends lots of requests against a web application, if the tested webserver is not designed to support high traffic loads, it might crash;
- we do not run into legal issues: a bruteforce directory listing might be tagged as a bruteforce attack. Dirbusting is an inner part of Penetration Testing. As such, it should be regulated by contracts.

As described in Section 1, we define a *"valid request"* as an HTTP request that has revealed a new path within the target website. We compare the total number of executed HTTP requests with the number of valid HTTP requests. To improve the significance of the results, we have repeated each experiment 30 times for each website.

With the bruteforce approach, the unique wordlist is shuffled, and words are used to perform the classic dirbusting procedure. On the other hand, with the clustering approach, we use the algorithm we have described in the previous section to select the words from the wordlist in a non-random way.

We compared the results graphically by plotting the relationship between the total requests sent to the target website and the number of valid requests.

Our goal is to compare the above-mentioned approaches by measuring the overall number of requests sent to the web application in order to complete the discovery of a website's structure thoroughly. Such a comparative evaluation aims to show that performance increases when using the semantic clustering approach.

5.1 Experiment information

The following information is useful to describe the performed experiments better. Firstly, we do not evaluate the time required to complete the task. Instead, we aim to verify that our solution reduces the number of HTTP requests sent to reconstruct the entire structure of the target website. Hence, we do not compare the execution time of the two approaches.

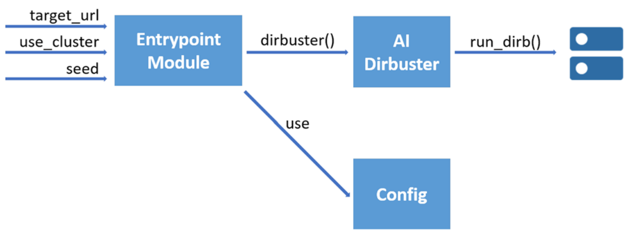


Fig. 5 Architecture of the dirbusting tool used to perform experiments. The input parameters are the target URL, a boolean value that enables the semantic clustering approach, and a seed value used to increase the randomness of the wordlist's shuffling. The tool is instrumented through a configuration module, while the HTTP requests are performed through the dirbuster module

As already anticipated, we have created an isolated environment by using container-based virtualization. No highly-intensive processes have been run during the tests. We continuously monitored CPU, RAM, and disk usage during each experiment to ensure that none went under pressure during any of the trials. We also verified that the tool did not crash during any of the runs.

Network issues might impact the results of the campaign. While we are not specifically focusing on response times, a stable network is crucial. Network problems could lead to response timeouts, potentially invalidating results. For these reasons, websites are deployed in an isolated Docker network. The environment operates within the context of the dirbusting instrumented tool. Therefore, we can safely assume that there are no network reliability issues during the experiments.

5.2 Architecture of the benchmarking tool

To perform experiments, we instrumented a dirbusting tool that is illustrated in Figure 5

The tool in question was developed in Python 3.6 and is made of three components:

- *Entrypoint*: accepts input parameters needed to set up a trial associated with a specific target;
- *AiDirBuster*: dirbusting module that implements the dirbusting process either in bruteforce mode or by leveraging the semantic clustering strategy proposed by us;
- *Config*: configuration module containing several configuration parameters, such as groups of words identified through semantic clustering.

The tool accepts the following parameters as inputs:

- *use_clustering*: a boolean value. If true, dirbusting uses the semantic clustering strategy; otherwise, a brute-force approach is adopted;

- *target_url*: the target web application used to run the experiment;
- *seed*: a seed used to increase randomness when the wordlist is shuffled during the brute-force approach.

Semantic clusters are computed offline and subsequently stored in the above-mentioned *Config* module.

Universal Sentence Encoder (USE) [9], version 4[1], implemented in TensorFlow 2.2.0 [39], is used to extract sentence embeddings.

We leveraged the K-means clustering algorithm implementation made available by the *sklearn* python library to find and collect clusters. The default set of parameter values was used, except for the factor 'K', which was properly configured with the elbow method.

5.3 Experimental environment setup

To simulate the dirbusting process, we have built a docker environment composed of 8 publicly available web applications, some of which are also typically used for experimenting with vulnerability assessment and penetration testing.

Table 3 shows the characteristics of the web applications in question.

Among the applications reported in the table, the ones that are usually used to experiment with web application penetration testing are Bodgeit,[2] bricks,[3] DVWS[4] (Damn Vulnerable Web Services), XVWA[5] (Xtreme Vulnerable Web Application) and Wacko.[6]

On the other hand, Wordpress,[7] Drupal[8] and Joomla[9] are among the most widely spread PHP Content Management Systems used to create web applications.

The environment realized for the experiment is built by using an Infrastructure as Code (IaC) approach. A docker-compose file including eight services describes the system's architecture under test, as shown in Fig. 6. Each service exposes the standard HTTP service (port 80) and maps it onto an unassigned TCP port of the hosting machine. The semantic clustering dirbusting tool sends requests to the eight web applications by targeting such exposed TCP ports on the host. With this approach, it is possible to add new web applications in an easy way, as well as to extend the experiment by including new target applications.

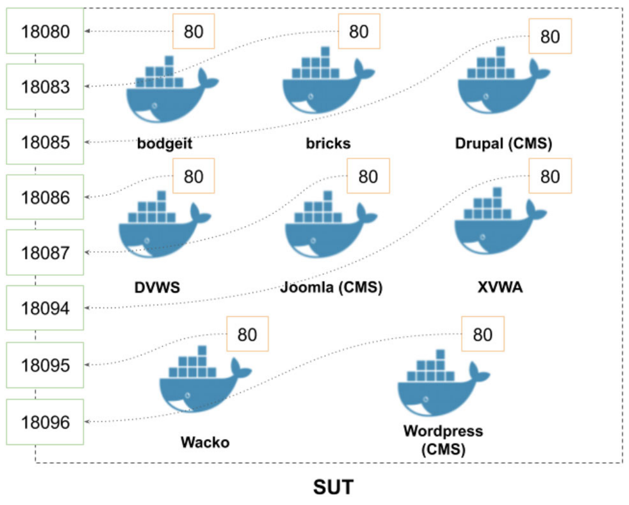


Fig. 6 The container-based testbed architecture used to evaluate the proposed algorithm. Each web application runs inside a Docker container and exposes an unassigned TCP port in the host machine

5.4 Wordlist acquisition

We extracted paths from each webserver to create the integrated wordlist and merged them in a single file. Given n the number of webservers used for the experiments, the following formula applies:

$$UniqueWordlist = rand(\sum_{w=1}^{n} absolute\_paths_w)$$

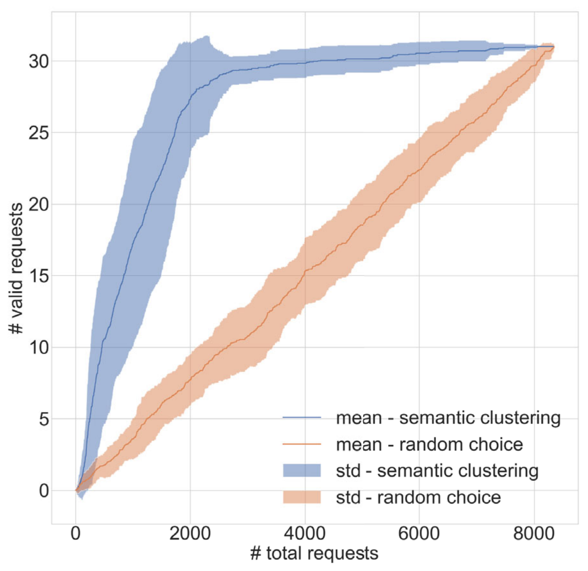
In a nutshell, *UniqueWordlist* can be obtained as the randomized concatenation of all absolute paths contained in each webserver.

The path extraction task for a specified webserver can be carried out by executing OS-level commands inside the related Docker service. The following one-line command is a practical example of how the above-mentioned task might be completed:

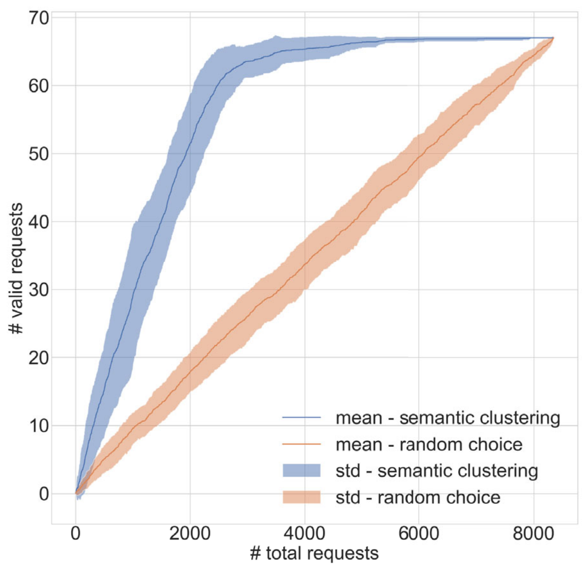
```
docker exec -it <webserver> bash
cd /var/www
find .  | sed 's/^\.//g'
```

The concatenation of all of the collected words creates a unified wordlist. As described in the previous section, the words in the wordlist are basically absolute paths. For our experiments, the final unified wordlist is composed of 8367 words.

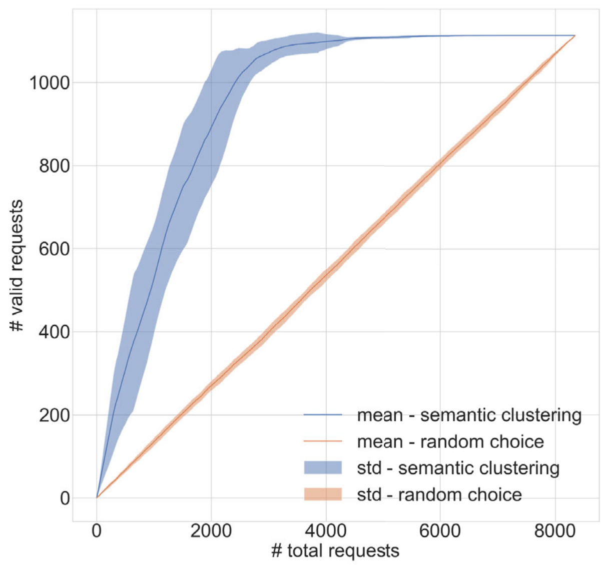
[1] https://tfhub.dev/google/universal-sentence-encoder/4
[2] https://github.com/psiinon/bodgeit
[3] https://sourceforge.net/projects/owaspbricks/
[4] https://github.com/snoopysecurity/dvws
[5] https://github.com/s4n7h0/xvwa
[6] https://github.com/adamdoupe/WackoPicko
[7] https://wordpress.com/
[8] https://www.drupal.org/
[9] https://www.joomla.org/



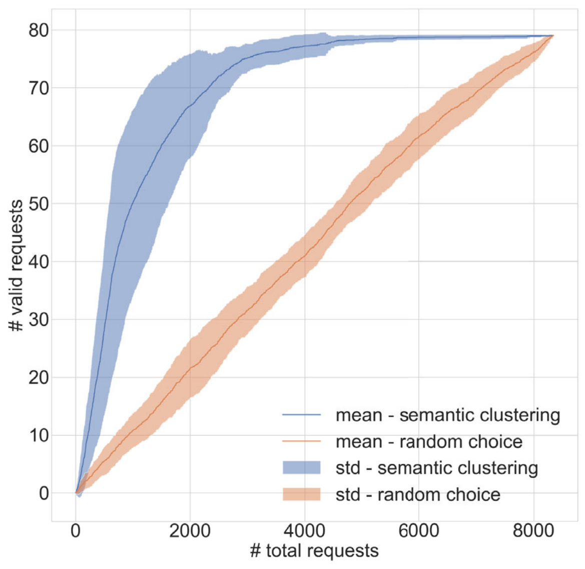
(a) Bodgeit



(b) Bricks



(c) Drupal



(d) DVWS

Fig. 7 Performance plots for Bodgeit, Bricks, Drupal, and DVWS web applications. The plots depict the mean and standard deviation trends of the detected valid requests relative to the total requests sent to the target server, averaged over 30 experiments. As observed, the semantic clustering approach (blue) demonstrates a performance improvement of nearly 50% compared to the legacy brute-force strategy (orange)

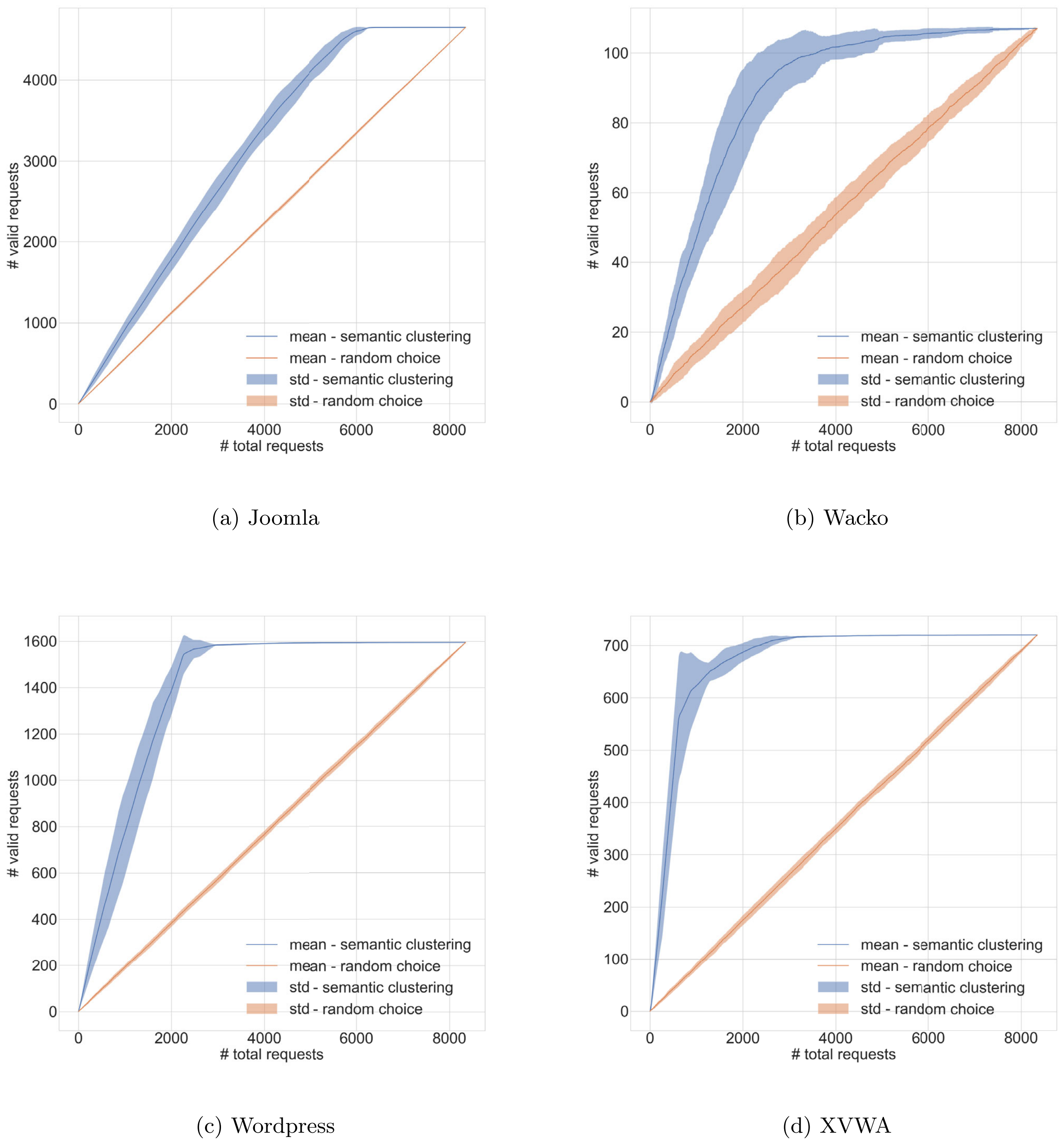


Fig. 8 Performance plots for Joomla, Wacko, Wordpress, and XVWA web applications. The results mirror those in Fig. 7, with the exception of Joomla, where the performance improvements are less pronounced. This is attributed to the fact that the wordlist used for the experiment contains more than half of the words related to Joomla. Hence, selecting a comprehensive wordlist is crucial for achieving favorable results

Table 3 Web applications used for the experiment

Web Application	Language Extension	path name convention
Bodgeit	jsp	under case
Bricks	php	camel case
Drupal (CMS)	php	snake case
DVWS	php	snake case
Joomla (CMS)	php	upper case
Wacko	php	snake case
Wordpress (CMS)	php	kebab case
XVWA	php	snake case

6 Experimental results

Our experiments allowed us to demonstrate that the enhanced dirbusting strategy we propose outperforms the legacy brute-force approach. Indeed, for each of the eight web servers under test, we achieved a performance improvement of up to 50%.

In Fig. 7 and Fig. 8, we show the results of our campaign. For each web server, we plot both the mean and the standard deviation (*std*) trend of the detected valid requests over the number of total requests addressed to the target server. Each experiment has been replicated 30 times in order to improve the significance of the collected results.

As we anticipated above, a "valid request" is a request with a response code other than 404 (*Not Found* HTTP error message).

In each of the plots, the two approaches are compared. In orange, we show the results of the legacy random brute-force approach, while in blue, we report the performance of the proposed semantic clustering strategy.

As it is possible to observe, the random brute-force strategy shows a linear trend. Indeed, as we apply randomization for each experiment, the number of requests needed to find all the paths is, on average, equal to the number of paths. In this way, the longer the wordlist, the higher will be the number of requests required to find the valid paths.

On the other hand, semantic clustering shows a steeper growth rate. As a matter of fact, with this approach, the curve stops increasing much earlier than with the brute force one.

With all web servers under test, the trend of the two approaches always remains the same. This indicates the gain in performance that can be achieved by leveraging the proposed semantic clustering approach.

Furthermore, our approach can identify almost all available target URLs with only half of the requests compared to the brute-force approach. This results in a performance improvement of close to 50%.

The only exception is Joomla, where the performance improvements are comparatively lower than those observed in the other web applications under test. The reason behind this finding is that the number of words collected for Joomla is 4672, which is more than half the total number of words included in our wordlist. This is evident when looking at the word count in Table 2. As observed, Joomla covers about half of the integrated wordlist.. This means that with a random approach, we can find valid URLs in Joomla with a probability of about 50%, reducing the potential gain achievable by leveraging the alternative approach we propose. Nevertheless, there is still a clear improvement for this web application as well, with around 2000 requests less than those needed to find all of the available paths with the brute-force approach.

7 Discussion

Many studies explore security testing in the web application domain, but current works often neglect the problem of optimizing the enumeration phase. The primary objective of our work has been to fill this gap. Specifically, we explore a novel approach based on semantic clustering to optimize the "dirbusting" technique, which is one of the most common approaches used to discover the structure of web applications during the enumeration phase of web application penetration testing.

This technique employs a "brute-forcing" fuzz of known path names to uncover hidden folders and files within the web application. The approach is well-defined in business security standards, such as Common Weakness Enumeration (CWE) [40] and Common Attack Pattern Enumeration and Classification (CAPEC) [41] methodology. Additionally, it can be instrumental in identifying security disclosure flaws, as the discovered paths may contain hidden sensitive data that could be exposed to malicious attackers.

Regrettably, conventional approaches are typically time-consuming, involving security experts who manually inspect the components of the web application and attempt to guess potential valid paths.

We demonstrate the feasibility of optimizing this activity through a semantic clustering approach. Our experiments indicate that the total number of HTTP requests needed to uncover the structure of a web application can be reduced by up to 50% compared to the classic dirbusting approach.

It is important to note that performance is highly dependent on the chosen wordlist. In our experiment, where the wordlist consists of approximately 50% Joomla paths, the approach does not yield significant benefits. Therefore, it is crucial in the initial setup to properly configure the solution with a comprehensive wordlist.

It is crucial to emphasize that this study serves as an introduction to the effectiveness of semantic clustering in addressing the challenge of web application structure discovery. The primary goal was to demonstrate that a semantic clustering approach can be applied in the unexplored realm of the enumeration phase, optimizing the typically manual dirbusting process. As such, trials and experiments were conducted in a controlled, small, and isolated environment, primarily focused on vulnerable applications and Content Management Systems.

Further studies should be undertaken to assess the solution with more comprehensive wordlists, diverse features, and a broader range of applications.

This assumption is valid, as the main goal of a penetration tester activity is to completely discover web application vulnerabilities.

The proposed approach should be further analyzed in production environments. In real-world scenarios, web applications are often deployed behind web application firewalls that could potentially block certain HTTP requests, impacting the effectiveness of dirbusting. Bypass techniques for such firewalls are an important consideration but are outside the scope of this work.

For the purposes of this study, the assumption is made that the security tester is authorized to perform the assessment, and defense firewalls are disabled. This assumption aligns with the primary goal of penetration testing, which is to thoroughly discover vulnerabilities in web applications.

The proposed ideas are generalizable and can be adapted to specific needs. Future work could involve extending experiments to include real-world applications for a more comprehensive evaluation. Additionally, exploring the application of large language models (LLMs) could be beneficial to identify further improvements to the approach. Combining different techniques, such as standard web spidering with dirbusting, is another avenue for improvement. This could involve using spidering to detect the overall structure of the target web application and then leveraging dirbusting to uncover hidden or private pages.

It is important to note our work focuses on demonstrating the effectiveness of a semantic clustering approach compared to a brute-force one for discovering the web application structure. As a result, strategies for exploring subpaths are not considered in this study. The wordlists used during the experimentation consist of full paths (e.g., */users/mooney/ir-course/*). However, dirbusting techniques can be recursive, meaning that whenever a new path is found, dirbusting can be recursively applied to it to discover new subpaths. This recursive aspect could be explored in future work for a more comprehensive analysis. In view of the above considerations, it would be interesting to investigate the use of our semantic clustering approach in a recursive way while also evaluating the possible alternative strategies for applying recursion while navigating through the dynamically identified subpaths (e.g., breadth-first, depth-first, etc.).

8 Conclusions

Web application security testing is paramount for shielding digital assets against potential threats. Security testers, pivotal in identifying and addressing vulnerabilities, often engage in directory busting during the enumeration phase. In this study, we introduce "dirclustering", a semantic clustering approach, as a significant optimization to traditional dirbusting techniques. Through meticulous experimentation in a controlled testbed, our results demonstrate the superior effectiveness of "dirclustering" compared to conventional dirbusting methods. This work opens a promising avenue for enhancing the enumeration phase of web application security testing, providing valuable insights for security testers to optimize their processes and achieve more efficient results in subsequent assessment phases.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Williams, J.: The OWASP project (ed.) (no date) Testing guide 4 - OWASP. https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf. Accessed 04 Feb 2024
2. Ibrahim, R., Zeebaree, S., Jacksi, K.: Survey on semantic similarity based on document clustering. Adv. Sci. Technol. Eng. Syst. J **4**(5), 115–122 (2019)
3. Naik, M.P., Prajapati, H.B., Dabhi, V.K.: A survey on semantic document clustering. In: 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), pp. 1–10 (2015). IEEE
4. Fellbaum, C.: Wordnet. The encyclopedia of applied linguistics. Chichester, England: Wiley-Blackwel (2012)
5. mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
6. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. Adv. Neural Inf. Process. Syst. **26**, 3111–3119 (2013)
7. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference

on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543 (2014)
8. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. Int. Conf. Mach. Learn. **32**, 1188–1196 (2014)
9. Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al.: Universal sentence encoder. arXiv preprint arXiv:1803.11175 (2018)
10. Conneau, A., Kiela, D., Schwenk, H., Barrault, L., Bordes, A.: Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364 (2017)
11. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
12. Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., Specia, L.: SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In: Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), pp. 1–14. Association for Computational Linguistics, Vancouver, Canada (2017). https://doi.org/10.18653/v1/S17-2001 . https://www.aclweb.org/anthology/S17-2001
13. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recognit. Lett. **31**(8), 651–666 (2010)
14. Song, X., Wu, M., Jermaine, C., Ranka, S.: Conditional anomaly detection. IEEE Trans. Knowl. Data Eng. **19**(5), 631–645 (2007). https://doi.org/10.1109/tkde.2007.1009
15. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Network anomaly detection: methods, systems and tools. IEEE Commun. Surv. Tutor. **16**(1), 303–336 (2014). https://doi.org/10.1109/SURV.2013.052213.00046
16. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. **10**(1145/1541880), 1541882 (2009)
17. Ahmed, M., Naser Mahmood, A., Hu, J.: A survey of network anomaly detection techniques. J. Netw. Comput. Appl. **60**, 19–31 (2016). https://doi.org/10.1016/j.jnca.2015.11.016
18. Pang, G., Shen, C., Cao, L., Hengel, A.V.D.: Deep learning for anomaly detection: a review. ACM Comput. Surv. (2021). https://doi.org/10.1145/3439950
19. Aslan, A., Samet, R.: A comprehensive review on malware detection approaches. IEEE Access **8**, 6249–6271 (2020). https://doi.org/10.1109/ACCESS.2019.2963724
20. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. ACM Comput. Surv. (2017). https://doi.org/10.1145/3073559
21. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y.: A survey of android malware detection with deep neural models. ACM Comput. Surv. (2020). https://doi.org/10.1145/3417978
22. Deore, M., Kulkarni, U.: Mdfrcnn: malware detection using faster region proposals convolution neural network. Int. J. Interact. Multimed. Artif. Intell. **7**(4), 146 (2022). https://doi.org/10.9781/ijimai.2021.09.005
23. Karbab, E.B., Debbabi, M.: Maldy: portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports. Digital Invest. **28**, 77–87 (2019)
24. Malhotra, R., Chug, A., Hayrapetian, A., Raje, R.: Analyzing and evaluating security features in software requirements. In: 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), pp. 26–30 (2016). IEEE
25. Godefroid, P., Peleg, H., Singh, R.: Learn&fuzz: Machine learning for input fuzzing. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 50–59 (2017). IEEE
26. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: Passgan: A deep learning approach for password guessing. In: International Conference on Applied Cryptography and Network Security, pp. 217–237 (2019). Springer
27. Nourashrafeddin, S., Milios, E., Arnold, D.V.: An ensemble approach for text document clustering using wikipedia concepts. In: Proceedings of the 2014 ACM Symposium on Document Engineering, pp. 107–116 (2014)
28. Wu, Z., Zhu, H., Li, G., Cui, Z., Huang, H., Li, J., Chen, E., Xu, G.: An efficient Wikipedia semantic matching approach to text document classification. Inf. Sci. **393**, 15–28 (2017)
29. Desai, S.S., Laxminarayana, J.: Wordnet and semantic similarity based approach for document clustering. In: 2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), pp. 312–317 (2016). IEEE
30. Sahni, L., Sehgal, A., Kochar, S., Ahmad, F., Ahmad, T.: A novel approach to find semantic similarity measure between words. In: 2014 2nd International Symposium on Computational and Business Intelligence, pp. 89–92 (2014). IEEE
31. Wei, T., Lu, Y., Chang, H., Zhou, Q., Bao, X.: A semantic approach for text clustering using wordnet and lexical chains. Expert Syst. Appl. **42**(4), 2264–2275 (2015)
32. Fiorini, N., Harispe, S., Ranwez, S., Montmain, J., Ranwez, V.: Fast and reliable inference of semantic clusters. Knowl. Based Syst. **111**, 133–143 (2016)
33. Zhang, L., Li, J., Wang, C.: Automatic synonym extraction using word2vec and spectral clustering. In: 2017 36th Chinese Control Conference (CCC), pp. 5629–5632 (2017). IEEE
34. Li, C., Lu, Y., Wu, J., Zhang, Y., Xia, Z., Wang, T., Yu, D., Chen, X., Liu, P., Guo, J.: Lda meets word2vec: a novel model for academic abstract clustering. In: Companion Proceedings of the The Web Conference 2018, pp. 1699–1706 (2018)
35. Alshari, E.M., Azman, A., Doraisamy, S., Mustapha, N., Alkeshr, M.: Improvement of sentiment analysis based on clustering of word2vec features. In: 2017 28th International Workshop on Database and Expert Systems Applications (DEXA), pp. 123–126 (2017). IEEE
36. Karagkiozis, N.: Clustering Semantically Related Questions Örebro University, School of Science and Technology. (2019)
37. Asgari-Chenaghlu, M., Nikzad-Khasmakhi, N., Minaee, S.: Covid-transformer: Detecting trending topics on twitter using universal sentence encoder. arXiv preprint arXiv:2009.03947 (2020)
38. Bodrunova, S.S., Orekhov, A.V., Blekanov, I.S., Lyudkevich, N.S., Tarasov, N.A.: Topic detection based on sentence embeddings and agglomerative clustering with Markov moment. Future Internet **12**(9), 144 (2020)
39. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O.,Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org (2015). https://www.tensorflow.org/
40. Corporation, M.: CWE-552: Files or directories accessible to external parties. https://cwe.mitre.org/data/definitions/552.html (2023)
41. Corporation, M.: CAPEC-143: Detect unpublicized web pages. https://capec.mitre.org/data/definitions/143.html (2023)