The Journal of Systems & Software 190 (2022) 111322

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Online malicious domain name detection with partial labels for large-scale dependable systems☆

Yongqian Sun [a,b,c], Kunlin Jian [a], Liyue Cui [a], Guifei Jiang [a], Shenglin Zhang [a,b,c,*], Yuzhi Zhang [a,b,c], Dan Pei [d]

[a] College of Software, Nankai University, Tianjin, 300457, China
[b] Tianjin Key Laboratory of Operating System, Tianjin, 300450, China
[c] Haihe Laboratory of Information Technology Application Innovation, Tianjin, 300450, China
[d] Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China

ARTICLE INFO

Article history:
Received 1 October 2021
Received in revised form 12 February 2022
Accepted 26 March 2022
Available online 9 April 2022

Keywords:
DGA
DNS
Feature selection
PU learning
Reinforcement learning

ABSTRACT

Detecting malicious non-existent domain names (NXDomains) in a real-time manner is vitally important to the security of large-scale dependable systems. Existing detection methods are trained based on the assumption that the NXDomains, which cannot be recognized by the domain generation algorithm (DGA) archive, are benign. However, new types of malicious NXDomains are continuously generated, and the DGA archive cannot cover all of them, making the NXDomains partially labeled. Additionally, extracting all the features for distinguishing malicious and benign NXDomains is computationally inefficient and inappropriate for online detection in large-scale dependable systems. This work proposes a framework, PUFS, to train an accurate malicious NXDomain detection model according to partial labels and conduct efficient online detection for large-scale dependable systems. PUFS adopts a novel, simple, yet effective three-step strategy to combine PU learning and feature selection. We conduct extensive experiments using real-world data collected from a top-tier global online bank. PUFS achieves 99.19% of F1-Score, and improves the feature extraction efficiency by 1153%, making it suitable for online detection scenarios.

1. Introduction

The security of large-scale dependable systems, such as online banks, online shopping services, is vitally important to users because a security problem of these systems can bring enormous economic loss. Among the security threats to large-scale dependable systems, domain name system (DNS) botnet is an important one, which has become increasingly popular in recent years (Schüppen et al., 2018; Khormali et al., 2020; Plohmann et al., 2016; Zhauniarovich et al., 2018; Bilge et al., 2011; Antonakakis et al., 2012; Schiavoni et al., 2014; Wang et al., 2017; Truong and Cheng, 2016; Tong et al., 2019). With the botnet, a botmaster (attacker) uses command and control (C&C) channels to manipulate devices infected by malware to perform attacks. A domain generation algorithm (DGA), which randomly generates a large number of domain names, is one of the most famous techniques used by botmasters, and its working principle is shown in Fig. 1. When a botmaster plans to perform an attack, it runs a collection of DGAs to generate domain names, randomly selects one or more from these domain names, and registers them in DNS servers. The registered domain names are directed to the C&C server. After that, a bot (victim) runs the built-in DGAs contained in the malware and tries to access the newly created domain names one by one. Once a domain name is successfully queried, the bot successfully finds the C&C server. The botmaster then communicates with the infected bot through the C&C server and starts its malicious activities.

To prevent the C&C server from being easily discovered and blocked, a bot's malware usually generates a large number of domain names using DGAs, few of which are registered as valid domain names (DGA, 2022). In this way, many non-existent domains (NXDomains) are produced. Therefore, we can detect DGA-generated malicious domain names by analyzing the patterns of NXDomains (Schüppen et al., 2018; Tong et al., 2019). However, a user can trigger a non-existent domain name response due to many other reasons. For example, a user may access a non-existent domain name due to a spelling error. Additionally, a certain application may generate NXDomain records because of misconfigurations. In other words, the NXDomains can be classified into two types: malicious algorithmically-generated domains (mAGDs) generated by DGAs and benign non-existent domains

☆ Editor: W. Eric Wong.

* Corresponding author at: College of Software, Nankai University, Tianjin, 300457, China.
E-mail address: zhangsl@nankai.edu.cn (S. Zhang).

https://doi.org/10.1016/j.jss.2022.111322
0164-1212/

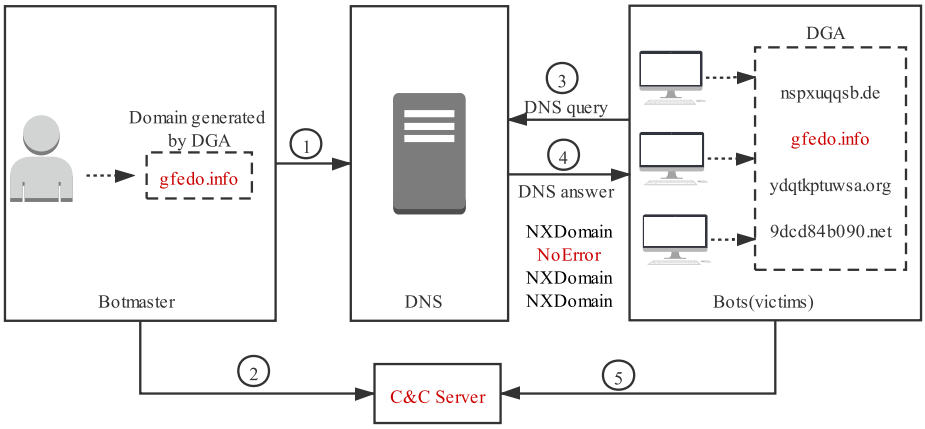


Fig. 1. The working principle of DGA.

(bNXDs) produced by spelling error, misconfiguration, etc. Therefore, it is of great importance to distinguish mAGDs from bNXDs. Considering the large scale of NXDomains, we attempt to take advantage of machine learning approaches to automatically detect mAGDs.

We can quickly obtain a large number of mAGDs generated by diverse types of DGAs from DGArchive (Plohmann et al., 2016). However, the DGArchive does not cover all mAGDs, and an enormous number of new mAGDs are continuously generated by new types of DGAs. Therefore, the domain names collected from large-scale dependable systems, which can be used to constitute the training set, include labeled (appear in DGArchive) mAGDs, unlabeled (not appear in DGArchive) mAGDs, and unlabeled bNXDs. Consequently, we cannot apply supervised learning methods, requiring that all positive samples (mAGDs) and negative samples (bNXDs) in the training set are labeled, for mAGDs detection. Naturally, positive-unlabeled learning (PU learning) (Bekker and Davis, 2020), which trains a classifier using positive and unlabeled samples, comes into our sight.

A machine learning method usually extracts the features of domain names and train a classifier for mAGDs detection. Previous researches have concluded 44 features to distinguish mAGDs from bNXDs (see Section 2.1 for details). Usually, the more features a machine learning method applies, the higher accuracy it achieves. However, extracting all the 44 features makes the online detection of mAGDs computationally inefficient. Specifically, hundreds of thousands to millions of NXDomains are generated every minute in large-scale dependable systems. Extracting all the 44 features for 1,000,000 NXDomains within one minute will "consume" 54 machines, which makes the machine learning method inapplicable in practice (operators are not willing to use too much computational resources for mAGD detection). Due to the diversity of NXDomains, randomly selecting a fraction of features for model training will significantly degrade the performance of machine learning methods. Generally, automatic feature selection can be applied to improve the efficiency of feature extraction (Deng et al., 2019), and we thus also use it in our scenario.

However, the combination of PU learning and feature selection brings an excellent challenge for mAGD detection. If we first perform PU learning and then conduct feature selection in offline training, the input of the trained classifier covers all the 44 features. In this way, we still have to extract all the 44 features in online detection for the trained classifier, and feature selection does not work in this case. Consequently, we should carry out feature selection first and then apply PU learning to train a classifier. Because supervised feature selection methods are usually more accurate than unsupervised feature selection methods (Deng et al., 2019), we apply them in our work. However, they require that all the samples in the training set are fully labeled, which is inconsistent with our scenario where the samples are partially labeled. Therefore, we should address this challenge before we integrate PU learning with feature selection.

In this paper, we propose PUFS, integrating PU learning with feature selection, to accurately and efficiently detect mAGDs based on partially labeled samples for large-scale dependable systems. PUFS applies a novel, simple, yet effective three-step strategy combining reliable negative (RN) extraction, feature selection, and classifier training. The combination of RN extraction and classifier training achieves PU learning, i.e., learning the patterns of mAGDs from partial labels. Additionally, PUFS applies a multi-objective reinforcement learning-based method to achieve optimal feature selection. To address the challenge imposed by integrating feature selection and PU learning, we first extract reliable negative samples to initialize feature selection. After that, we iteratively update feature selection, RN extraction, and classifier training until we obtain the optimal feature set an accurate classifier.

The main contributions of this work can be summarized as follows:

- To the best of our knowledge, we are among the first to identify the partially labeling problem and the inefficient feature extraction problem in training an mAGD detection model for large-scale dependable systems.
- We propose a novel three-step strategy to address the challenge introduced by integrating PU learning with feature selection. This strategy can be applied to more scenarios beyond automatic mAGD detection.
- To comprehensively evaluate the performance of PUFS, we conduct extensive evaluation experiments using data collected from a top-tier global online bank, which is a large-scale dependable system. PUFS achieves 99.19% of F1-Score and improves the feature extraction efficiency by 11.53 times compared to a state-of-the-art method, making it suitable for online mAGD detection scenarios.

The rest of this paper is organized as follows. We introduce the background of this paper, include domain name features, PU learning, and feature selection, in Section 2. The design of PUFS is elaborated in Section 3, followed by the description of evaluation experiments in Section 4. We discuss the related works in Section 5, and conclude our work in Section 6.

2. Background

In this section, we first present the domain name features that can be used for distinguishing mAGDs from NXDomains

Table 1
Extracted features.

#	Feature	Type
1	Domain name length	Integer
2	Number of subdomains	Integer
3	Longest label length	Integer
4	Subdomain length mean	Float
5	Ratio of hexadecimal parts	Float
6	Ratio of digit-exclusive subdomains	Float
7	Ratio of underscore	Float
8	Contains IPv4	Binary
9	Contains single-character subdomain	Binary
10	Contains TLD as subdomain	Binary
11	Has www prefix	Binary
12	Is exclusive prefix repetition	Binary
13	Alphabet size	Integer
14	Ratio of vowels	Float
15	Ratio of digits	Float
16	Ratio of repeated characters	Float
17	Ratio of consecutive digits	Float
18	Ratio of consecutive consonants	Float
19	Contains underscore	Binary
20	Contains digits	Binary
21	Digit letter count	Integer
22	Gibberish score	Float
23	Shannon entropy	Float
24	N-Gram of domain name	Vector

in Section 2.1. After that, we briefly introduce PU learning and feature selection in Section 2.2.

2.1. Domain character features

In this paper, we conduct mAGDs detection relying on domain names only. Therefore, we pay attention to the features of domain names that can distinguish mAGDs from bNXDs. These features have been well studied and proved to be practical in previous works. Usually, the more features an mAGD detection method uses, the more accurate it is. Consequently, we collect as many features as possible in our work, and a total of 24 features are collected. These features can be divided into three categories; namely, structural features, linguistic features, and statistical features (Schüppen et al., 2018; Tang et al., 2020). We list these features in Table 1. Among these features, the first 12 features are structural features, the middle eight (13–20) features are linguistic features, and the remaining four features are statistical.

Specifically, the domain names generated by DGAs are usually assigned a specialized length, so we use the length of each domain name as a feature (Antonakakis et al., 2012). Because mAGDs and bNXDs are typically different in the composition of characters (Ahluwalia et al., 2017), we calculate the ratio of numerical characters and vowel characters, respectively. Consecutive consonants and consecutive digits typically account for a relatively large proportion of malicious domain strings, and we thus apply this feature in mAGD detection (Schüppen et al., 2018). Since DGA-generated domains are usually random characters, entropy, which measures the randomness degree of domain name characters, can be used in our scenario (Antonakakis et al., 2012; Plohmann et al., 2016; Schüppen et al., 2018). Usually, mAGDs are not readable; therefore, the gibberish score, which measures how readable a string is, is applied in our work (Tang et al., 2020). Since the domain name-based mAGD detection problem can be represented as a text classification problem, N-gram, the well-known text classification technique, is also introduced. redFor a domain name, we split it into a contiguous sequence of N characters using the sliding window technique, which is called an N-gram. We adopt 1-gram, 2-gram, and 3-gram in our scenario, respectively. Then we calculate the frequency of occurrence of all sequences and get three frequency vectors. We calculate seven frequency distribution characteristics of these three vectors, including mean, standard deviation, median, 25% percentile, 75% percentile, minimum, and maximum. For example, for the domain name *earnestnessbiophysicalohax.com*, we remove the top-level domain and dot, and split it into 26 1-gram sequences, *i.e.*, $(e, a, r, n, e, s, t, n, e, s, s, b, i, o, p, h, y, s, i, c, a, l, o, h, a, x)$, with each sequence containing only one character. We count the frequency of occurrence of different sequences, *i.e.*, $[e, a, r, n, s, t, b, i, o, p, h, y, c, l, x]$, and obtain an array [3, 3, 1, 2, 4, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1]. Next, we calculate seven frequency distribution characteristics of this array, including mean (1.7), standard deviation (0.9), median (1.0), 25% percentile (1.0), 75% percentile (2.0), minimum (1), and maximum (4). These seven values constitute the 1-gram feature vector of the domain name. Similarly, we can calculate the 2-gram and 3-gram feature vectors, respectively. Therefore, the three N-gram feature vectors contain 21 values. Table 2 lists the 21 values in the N-gram feature vectors of the domain name *earnestnessbiophysicalohax.com*.

Table 2
The 21 values in the N-gram feature vectors of the domain name *arnestnessbiophysicalohax.com*.

#	Meaning	Value
1	1-gram mean	1.7
2	1-gram standard deviation	0.9
3	1-gram median	1.0
4	1-gram 25% percentile	1.0
5	1-gram 75% percentile	2.0
6	1-gram-minimum	1
7	1-gram maximum	4
8	2-gram mean	1.1
9	2-gram standard deviation	0.3
10	2-gram median	1.0
11	2-gram 25% percentile	1.0
12	2-gram 75% percentile	1.0
13	2-gram minimum	1
14	2-gram maximum	2
15	3-gram mean	1.0
16	3-gram standard deviation	0.2
17	3-gram median	1.0
18	3-gram 25% percentile	1.0
19	3-gram 75% percentile	1.0
20	3-gram minimum	1
21	3-gram maximum	2

2.2. PU learning and feature selection

Because NXDomains are very complicated, it is challenging, if not impossible, to label a vast number of bNXDs for training an accurate mAGD detection model in large-scale dependable systems. Additionally, with the help of DGArchive, we can only label a fraction of mAGD, and many mAGDs remain unlabeled since new mAGDs appear continuously. Accordingly, we must train an mAGD detection model based on partially labeled mAGDs (positive samples) and unlabeled bNXDs (negative samples). Intuitively, we can apply PU learning to address this problem (Li, 2013; Bekker and Davis, 2020). The assumptions of existing PU learning methods can be summarized from two perspectives: labeling mechanism and class distribution (Bekker and Davis, 2020). One can choose an appropriate PU learning method based on her scenario and the method's assumption. Previous studies on mAGD detection have demonstrated that domain name samples are separable, i.e., a domain name sample is either an mAGD or a bNXD (Chen et al., 2018; Schüppen et al., 2018). Therefore, we choose a two-step PU learning strategy, which is based on separability hypothesis. It consists of two main steps: extracting reliable negative samples from unlabeled samples and training a classifier.

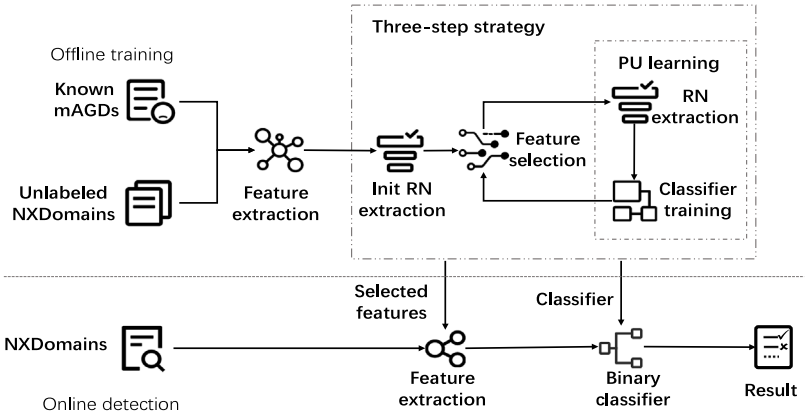


Fig. 2. The overview of PUFS.

Feature selection reduces the dimension of the original feature set, which is the most intuitive way to reduce feature extraction time (Deng et al., 2019). According to the way they combine with machine learning tasks, they can be divided into three categories: filter methods, wrapper methods, and embedded methods (Deng et al., 2019; Chandrashekar and Sahin, 2014). A filter method usually analyzes the correlation scores of features and selects the features with higher scores. For example, Ikram and Cherukuri (2016) and Song et al. (2010) applied principal component analysis (PCA) to extract principal components and calculated the participation degree of each feature. Giarelis et al. (2020) used the labels of samples to calculate the x-square score of features and sample labels. Typically, a filter method is computationally efficient, but it often ignores the dependencies of features and the interaction between features and classifiers and thus suffers from low accuracy (Song et al., 2010). Embedded methods, such as LASSO (Muthukrishnan and Rohini, 2016) and DT-RFE (Thakkar and Lohiya, 2021), incorporate feature selection as part of classifiers. They heavily rely on the performance of classifiers. If an inappropriate classifier is chosen, the performance of these methods will significantly degrade. A wrapper method takes the performance of the classifier as the optimization objective. For instance, AutoFS (Fan et al., 2020), a state-of-the-art wrapper method, uses reinforcement learning to iteratively search the feature space and determine the feature subset with which the classifier achieves the best performance. It applies multi-agent to reduce the action search space, which dramatically improves the training efficiency. In this work, we improve the performance of AutoFS, and the selected features simultaneously achieve high accuracy and efficient feature extraction.

3. Methodology

In this section, we first introduce the design overview of PUFS in Section 3.1, followed by the description of the main components, including feature extraction in Section 3.2, PU learning (including RN extraction and classifier training) in Section 3.3, and feature selection in Section 3.4. Finally, we explain how we integrate PU learning and feature selection into a novel, simple, yet effective three-step strategy in Section 3.5.

3.1. Design overview

We propose a novel framework, PUFS, to detect DGA-generated domain names from NXDomains. The overview framework is shown in Fig. 2. PUFS is mainly divided into two phases: the offline training phase and the online detection phase.

In the offline training phase, PUFS firstly executes data preprocessing to prepare the PU data for training. Next, feature extraction is adopted to extract all features from the PU data, which converts all the samples to feature vectors. After that, using the feature vectors, PUFS trains a three-step strategy model. This strategy implements a supervised feature selection method (i.e., TCFS) based on PU data, which reduces the time-consuming of online feature extraction. To obtain an optimized classifier, this strategy continuously iterates the three steps to optimize the model. After training, we get a set of selected features and a binary classifier for the online detection phase.

In the online detection phase, we first check whether there exist known mAGDs in NXDomains, and put them out if they exist. Otherwise, PUFS performs feature extraction according to the selected features, where the extraction time is far less than extracting all features. Then it detects the items of NXDomains to be positive (mAGD) or negative (bNXD) by using the trained binary classifier.

Note that the three-step strategy in offline training is the core idea of PUFS so that it will be elaborated in the following of this section.

3.2. Feature extraction

3.2.1. Data preprocessing

As mentioned in Section 1, the real-word training data includes two parts, positive set and unlabeled set. We collected a large number of NXDomains from a top-tier global online bank as the unlabeled set. Referring to some other works (Schüppen et al., 2018; Tang et al., 2020), we obtain DGArchive (Plohmann et al., 2016) as the positive set. Then we do some necessary data preprocessing before training: de-duplicating the dataset and removing the known mAGDs in NXDomians.

3.2.2. Feature analysis and design

In order to extract the characters of domain names more comprehensively, 44 features (shown in Table 1) are chosen after full investigation, as described in Section 2.1. Then through feature extraction, the domains of PU data are converted to many PU samples (i.e., positive samples and unlabeled samples), which are 44-dimensional feature vectors.

With these feature vectors, we could have trained a classifier by PU learning next. However, feature selection has to be carried out due to the time limitation in the online feature extraction, as described in Section 1.

3.3. PU learning

This subsection introduce the PU learning method of PUFS, which consists of two modules: RN extraction and classifier training. RN extraction is applied to extract reliable negative samples from the unlabeled samples. Then combined with the positive samples, we can train the classifier.

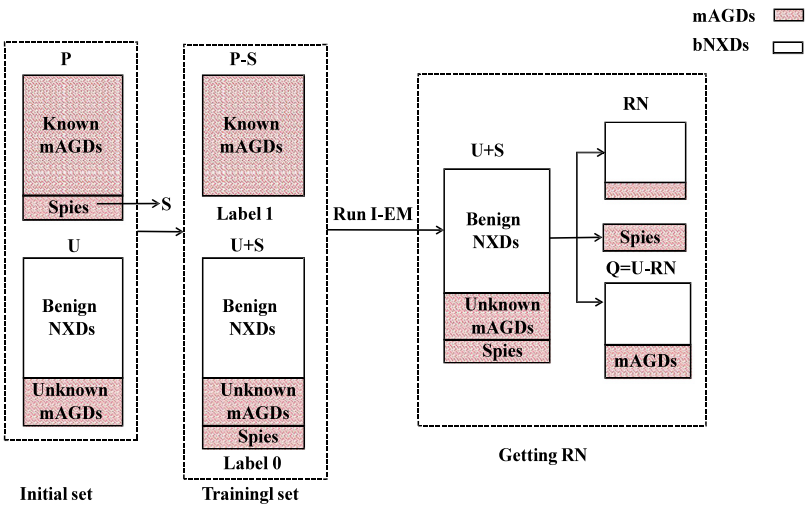


Fig. 3. Reliable negatives extraction of Spy.

3.3.1. RN extraction

There have been some studies on RN extraction, such as Spy (Liu et al., 2003), K-Means (Chaudhari and Shevade, 2012), and C–CRNE (Liu and Peng, 2014). We adopt the Spy method because of its better performance than others in this paper (will be shown in Section 4.3.1).

The Spy method is shown in Fig. 3. At first, we randomly take a proportion of s samples from the positive sample set (P) as the spy samples (S) and put them to the unlabeled sample set (U). Then, we get two new sample sets: P-S and U+S. Next, we run an I-EM algorithm (Liu et al., 2003) to train a classifier by using the new sets, and the pseudocode of I-EM is shown in Algorithm 1. In I-EM, we first label P-S and U+S as two classes c_1 and c_0 separately. Then we perform an iteration of a naive Bayesian classifier (NB-C) training: (1) We regard the samples with label c_1 as a positive set and c_0 as a negative set for training a naive Bayesian classifier; (2) Use the classifier to test and re-label the samples in U+S; (3) End the loop if all the sample labels in U+S are the same as the last iteration, otherwise goto (1). This way, we can get a final NB-C for specific initial sets of P and U, and obtain the RN set, which consists of the samples with c_0.

Algorithm 1: I-EM

input : positive samples: P-S, unlabeled samples with some spies: U+S
output: a naive Bayesian classifier NB-C

Label each sample d_i in P-S *the class* c_1;
Label each sample d_j in U+S *the class* c_0;
NB-C ← an initial naive Bayesian classifier built on U+S and P-S;
while *the class labels of* U+S *change* **do**
  for $d_j \in$ U+S **do**
    Re-label d_j using NB-C;
  end
  NB-C ← a new naive Bayesian classifier built;
end

Now we introduce the re-label process of each iteration in I-EM. We get a probability $Pr(d_j)$ when test a sample by a NB-C and then compare it with a threshold to determine the labels (i.e., if $Pr(d_j)$ is greater than the threshold label c_1, or label c_0). Then how to determine the threshold? We consider d_j in U as a reliable negative sample only when $Pr(d_j)$ is less than most spies' probabilities. Therefore, we set a parameter of proportion t, the corresponding probability $Pr(d_t)$ is regarded as the threshold, where d_t is the sample of the t quantile of the spies.

3.3.2. Classifier training

After RN extraction, we obtained the RN set and intended to train a binary classifier for online detection using P and RN. Therefore, a machine learning algorithm is required to train the classifier. In this paper, Random Forest (RF) is adopted because it is suitable for different types of features and has been proved to have a good performance in mAGDs detection case (Schüppen et al., 2018). Empirically, U usually contains few potential mAGDs, while the RN set above is generally much smaller than U (Bekker and Davis, 2020). So we should extract negative samples from Q (i.e., Q=U-RN) as much as possible so that RN can represent the characters of the negative set more accurately.

As shown in Fig. 4, we train the classifier by RF using an iteration way, which can constantly extract RN samples from Q. For each iteration i, the samples in Q_i are tested by classifier$_i$, and then the negative ones are added to RN, and the remaining constitute $Q_{(i+1)}$. It will keep iterating until $Q_{(i+1)}$=Q_i, and then we get the final classifier as the binary classifier.

3.4. Feature selection

The principal objective of feature selection is to improve feature extraction efficiency in the online detecting phase without degrading accuracy. Usually, a machine learning-based classifier, *e.g.*, decision tree, SVM, random forest, XGboost, requires that its input *in the training and testing set* have the same features (Lanzi, 2000). Therefore, to reduce the number of features needed to be extracted and improve the feature extraction efficiency for *online detection*, we should conduct feature selection, find the best feature subset, and reduce the number of features needed to be extracted *in offline training*. Recently, reinforcement learning-based methods have shown superior performance in feature selection (Liu et al., 2021) (more details can be seen in Section 4.3.2). Consequently, we add time constraint to an efficient and accurate reinforcement learning-based feature selection method, AutoFS (Fan et al., 2020), and name it TCFS. Its framework is shown in Fig. 5.

We use a multi-agent approach to reduce the dimension of action space, which creates an agent for each candidate feature. They chose actions (select or deselect the candidate feature) according to the advice given by their private policy network:

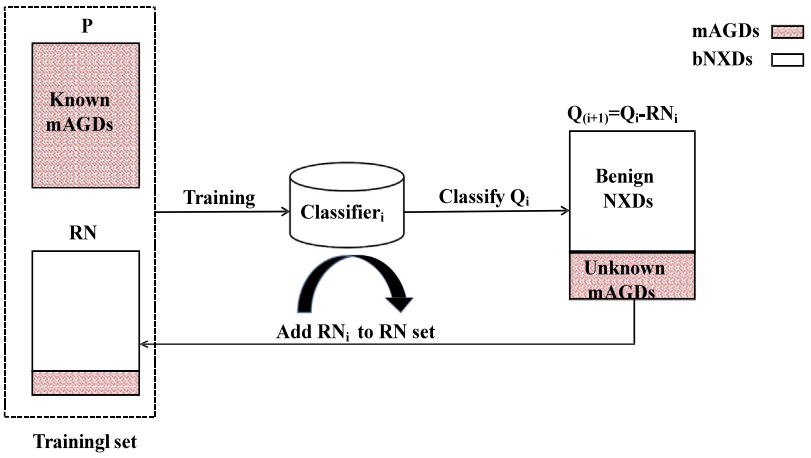


Fig. 4. Classifier training.

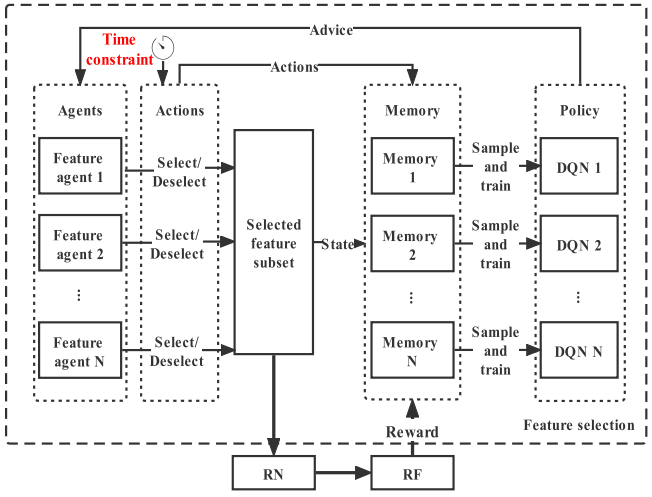


Fig. 5. The framework of TCFS.

Deep Q-Network (DQN) (Mnih et al., 2015). We perform a feature subset exploration step at each iteration, which contains two stages (control stage and training stage).

In the control stage, each agent takes the current state as the input and outputs the action suggested by their policy network. We use the method proposed by AutoFS to derive a comprehensive representation of the state with a graph convolutional network (GCN) (Bruna et al., 2013). It transforms the samples with changing size and content (because selected features changed) as a same-shape state vector, which ensures that the dimensions of input of DQN are consistent. At the same time, through RN extraction and RF classifier training, we will obtain an F1-Score that classifiers achieved to evaluate the next state. We will describe in detail how we get this F1-Score on PU data in Section 3.5. Next, the overall reward, which is defined as the weighted sum of the F1-Score, the redundancy, and the relevance of the selected feature subset quantified by the mutual information (Fan et al., 2020), is designated to all participating agents. For each agent, at time t, a tuple $(s_i^t, a_i^t, r_i^t, s_i^{t+1})$ will be stored in its memory, where s_i^t is the state, a_i^t is the action, r_i^t is the reward and s_i^{t+1} is the next state.

As shown in Algorithm 2, in the training stage, the policy network of each agent randomly derives a small batch of samples from the storage and trains the DQN network. When I iterations are completed or a satisfactory feature subset is explored, TCFS outputs the selected feature subset.

Algorithm 2: Feature selection with time constraint

input : number of iterations: I, number of features: N, time constrain: τ

Initialize state s, next state s', subset;
Initialize DQNs for N features: $\{d_1,d_2,...,d_N\}$;
Initialize actions for N features: $\{a_1,a_2,...,a_N\}$;
for $t \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** N **do**
    $a_i^t \leftarrow$ Select;
    subset $\leftarrow a_i^t$;
    if `Feature extraction time(subset)` $> \tau$ **then**
      $a_i^t \leftarrow$ Deselect;
    else
      $a_i^t \leftarrow$ chosen by d_i;
    end
    with probability ϵ reselect a random action a_i^t;
    subset $\leftarrow a_i^t$;
  end
  $\{r_1^t,r_2^t,...,r_N^t\}$, $s' \leftarrow$ in state s after actions $\{a_1^t,a_2^t,...,a_N^t\}$;
  $M \leftarrow (s, \{a_1^t,a_2^t,...,a_N^t\}, s', \{r_1^t,r_2^t,...,r_N^t\})$;
  $\{d_1^t,d_2^t,...,d_N^t\}$ are trained with samples from M;
  $s \leftarrow s'$;
end

As mentioned earlier, we should conduct feature selection *in offline training* to improve the feature extraction for *online detection*. Specifically, we try to use limited computational resources (say ten machines) to conduct online mAGDs detection for *online detection*. To achieve this goal, we set a time constraint τ, which indicates the maximum running time to extract the selected features for M NXDomains using one machine in *offline training stage* (and therefore in *online detection stage*). For example, when we set $\tau = 60$ and $M = 1,000,000$, we can extract features for $1,000,000$ NXDomains within one minute using one machine in *online detection stage*. Note that we set a time constraint instead of a number constraint on the selected feature subset because different features consume different feature extraction time, and a small number of features can consume a lot of feature extraction time. Accordingly, in each iteration, TCFS abandons a feature if this feature causes the extraction time to exceed τ.

Because AutoFS achieves high effectiveness and efficiency simultaneously, TCFS, which adds a time constraint to AutoFS, is also accurate and efficient in feature selection. Please note that

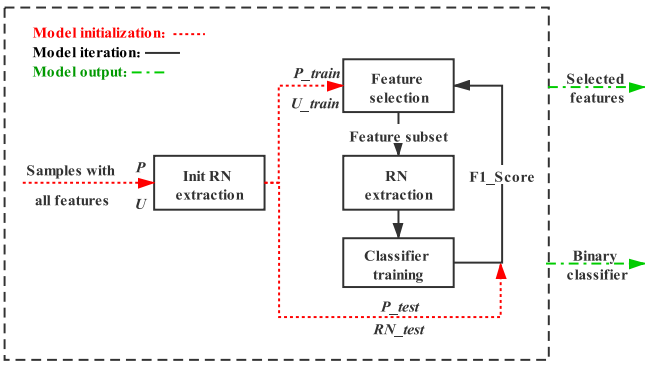


Fig. 6. The combination of feature selection and PU learning.

applying TCFS for feature selection is not the main contribution of this paper.

3.5. Three-step strategy

PUFS proposes a novel framework integrating PU learning and feature selection, as a three-step strategy. The detail is shown in Fig. 6.

For reinforcement learning methods, the external environment is usually required to provide an accurate reward value for each feature selection action. In the case of supervision, the samples with selected feature subset are used to training the classifier, then the accuracy it obtains in the test set is used as the reward value. However, with partial labels data, this reward value is not available. To solve this problem, we design a new reward value with the help of RN extraction and classifier training parts.

Firstly, we make an initialization. We perform RN-Extraction-module for initial RN based on all features from U. These samples in initial RN are considered as "true" negative samples in this strategy. Then combined with P samples, we can get the initial PU dataset. Next, we split the initial PU into two parts by a certain proportion (i.e., 8:2 in this paper): P_train, U_train for the model training iteratively, and P_test and RN_test for testing. We can calculate a value to evaluate the classifier's performance (we use F1-Score in our system). And this F1-Score is used to calculate the reward in reinforcement learning.

After initialization, the partially labeled feature selection problem can be solved in a supervised way. From the perspective of using reinforcement learning, we can provide a reasonable reward whenever the reinforcement learning algorithm takes action and goes to a state representing a feature subset. The reinforcement learning algorithm uses this reward value to update its action selection strategy and enter a new iteration. When the preset number of iterations (I) is completed or the feature subset satisfying the two target conditions (feature extraction time and F1-Score) is explored, the model outputs the selected feature subset and the binary classifier trained with samples on these features.

Because we use the F1-Score that our classifiers achieved on the data processed by the PU learning part, as one portion of the reward value, the classifier's performance will be regarded as one optimization objective of reinforcement learning feature selection. In this way, our three-step strategy makes PU learning and feature selection proceed simultaneously and finally outputs the selected feature subset and a trained binary classifier.

4. Evaluation

In this section, we conduct extensive experiments to demonstrate PUFS' performance. In Section 4.1, we introduce the datasets and experimental setup. Then we compare PUFS with baseline methods in Section 4.2 and replace each module of the three-step strategy with alternative methods to verify PUFS's performance in Section 4.3. Next, in Section 4.4, we try to remove RN extraction or feature selection models from PUFS to verify their importance. At last, we discuss how the hype parameters of PUFS affect its performance in Section 4.5 and show the result of feature selection in Section 4.6.

4.1. Experimental design

4.1.1. Dataset

We obtain the labeled positive data from DGArchive (Plohmann et al., 2016). Additionally, we also collect a large amount of real-world unlabeled data from a top-tier global online bank. To evaluate the trained classifier, we split out some data from the unlabeled data, and then label them manually and regard them as a ground truth set. Motivated by Tang et al.'s work (Tang et al., 2020), we invite three engineers to label the ground truth. Every engineer can label 5000–10,000 domain names per day. Due to the heavy workload, engineers first label a part of domain names and then summarize some rules. Then they use the rules to screen domain names. To ensure the label work more accurately, each domain name is labeled by at least two engineers independently. If there is a disagreement, the third engineer will be involved for re-checking.

The details of the three datasets are as follows:

- Positive data: We collect 63 million mAGDs of 107 DGA families from DGArchive in total.
- Unlabeled data: We collect massive records from the DNS resolver of a top-tier global online bank. These DNS logs are generated from October 1 2019 to October 30 2019. We extract 1.3 billion NXDomain responses from the 30-day records. In addition, we perform the de-duplication operation on these domain names. Then we delete the known mAGDs from these samples composed of known malicious DGA domains and get about 3.45 million unlabeled domain names.
- Ground Truth data: The labeled domain dataset used in Tang et al. (2020) contains 328 thousand bNXDs. Considering that mAGDs provided by DGArchive are relatively abundant, we sample 328 thousand mAGDs from DGArchive and used them as positive samples in Ground Truth. In contrast, the negative samples set of Ground Truth is served by the labeled bNXDs dataset.
- LOGO test: Motivated by Schüppen et al. (2018), we construct a more "clear" partially labeled set by injecting some mDGAs to the unlabeled data based on the idea of leave-one-group-out cross-validation (LOGO CV). In this work, the positive samples in this experiment consist of domain names of 66 DGA families we select from DGA-generated domain names data. We randomly extract 1000 domain names from each family. The LOGO test we designed contains a total of 66 trials. For each test, the training set includes 65 family domain names and the equivalent number of domain names extracted from unlabeled domain names data. We use one of all 66 DGA families as the test set for each test. Note that we take 20% of the domain name samples from the DGA family and add them to the unlabeled dataset of the training set. The remaining domain samples of the DGA family are used as the final test set. The specific experimental setup is shown in Fig. 7.

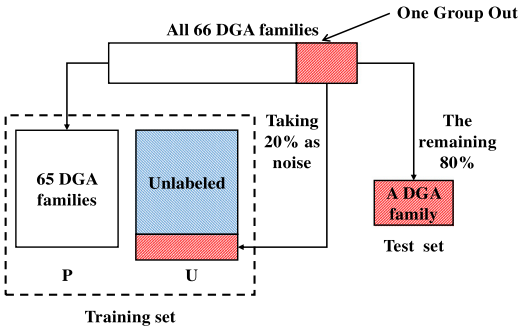


Fig. 7. Leave-One-Group-Out test. Successively take one DGA family as "One Group Out" which is unknown. Add 20% of it into unlabeled samples as unfilterable noise, and the remaining 80% is test set.

Table 3
Overall performance of PUFS and FANCI.

	Precision	Recall	F1-Score	FPR	FNR
FANCI	0.9959	0.9862	0.9911	0.00396	1.38e−2
PUFS	0.9841	0.9999	0.9919	0.00821	9.14e−6

4.1.2. Evaluation metrics

To evaluate the effectiveness of our model, we use the metrics of precision, recall and F1-Score, where $precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{TP+FN}$. F1-Score combines the characteristics of these two parameters: F1-Score $= \frac{2*precision*recall}{(precision+recall)}$. *TP* (True Positive) presents the number of positive samples that are classified as positive, and *FP* (False Positive) is the number of negative samples that are mistakenly classified as positive. *FN* (False Negative) is the number of positive examples that are predicted to be negative. *TN* (True Negative) means the number of negative examples that are predicted to be negative. In particular, False Positive Rate ($FPR = \frac{FP}{TN+FP}$) and False Negative Rate ($FNR = \frac{FN}{TP+FN}$) are also used to verify the effect of our model.

To evaluate model's efficiency, we propose a metric: number of machines, which refers to the number of machines required to perform the feature extraction of 1,000,000 NXDomains in one minute. It is given by $1,000,000/N$, where N refers to the number of NXDomains that one machine can carry out feature extraction processing in one minute.

4.1.3. Experimental environment

The evaluation experiments are implemented using Python 3.7.11, and run on an x64 server with 48 Intel Xeon E5-2650 2.20 GHz CPU and 128 GB RAM.

4.2. The overall performance of PUFS

In order to evaluate the performance of PUFS in a comparable way, we use FANCI as the baseline method (more details can be seen in Section 5). We show the performance of these models in the Ground Truth test in Table 3 and the number of machines used for feature extraction in Fig. 8. We also show the result of the LOGO test in Fig. 8.

As shown in Table 3 we can see that PUFS can achieve a high F1-Score similar to FANCI (0.9911). This shows that we keep an even detection performance when considering domain name detection as a PU scenario. And, obviously in Fig. 8, the FNR in the LOGO test of PUFS (0.021) is 30% smaller than FANCI (0.322). This proves that our system has an obvious advantage in terms of identifying unknown mAGDs in the PU scenario.

The detection efficiency of PUFS is significantly higher than the detection model without feature selection shown in Fig. 8,

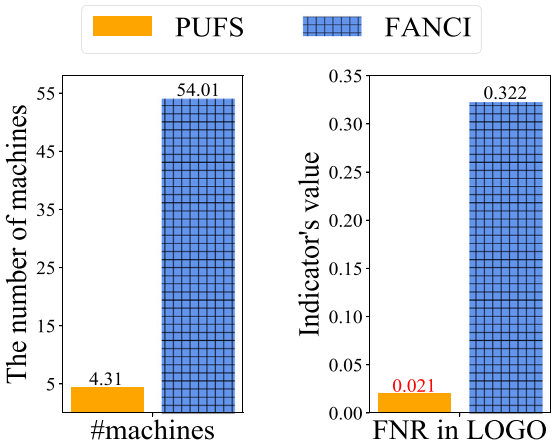


Fig. 8. Number of machines and FNR in LOGO test of PUFS and FANCI.

which proves that our work of combining feature selection with PU-learning is effective. Using PUFS, we significantly reduce the number of machines applied for feature extraction from 54 machines to 5 machines. Because of the substantial reduction of the processing time on feature extraction, PUFS is more competent for the task of online detection of large-scale dependable systems.

We acknowledge that it would require fewer machines if the methods were implemented using more efficient languages (e.g., C and C++) than using Python. However, many widely used machine learning models, including Random Forest and AutoFS, are implemented using Python. Therefore, applying Python to implement a machine learning-based mAGDs detection method is much more efficient, and both FANCI[1] and PUFS are implemented using Python. Moreover, PUFS is 11.5 times more efficient than FANCI in mAGDs detection, consuming 7.98% computational resources (*e.g.*, machines) compared to FANCI. Even if we implement PUFS and FANCI with other languages, the former will still be much more efficient than the latter. Usually, operators pay much attention to the efficiency of a mAGDs detection method. Accordingly, they prefer deploying PUFS to FANCI because it achieves the nearly as high accuracy with much fewer machines.

4.3. Alternative experiment

In this section, we test alternative basic methods for three modules of the three-step strategy: RN extraction, feature selection, and classifier training, and show their performance on the Ground Truth test.

4.3.1. Comparison among RN extraction methods

The RN extraction is the first part in the PUFS model, which is expected to find high-quality RN samples from unlabeled samples. In our malicious domain name scenario, only a small amount of positive samples are left in the U samples after being filtered by the blacklist. However, they may also seriously affect the classifier performance, which can be reflected in the LOGO test. We present the Ground Truth test performance of PUFS with (w) different RN extraction methods:

(a) K-Means. It simply clusters all samples into two clusters using K-Means, and takes the samples furthest from the positive cluster as RN.

[1] https://github.com/fanci-dga-detection/fanci.

Table 4
Comparison among RN extraction methods.

	Precision	Recall	F1-Score	FPR	FNR
w K-Means	0.5525	1.0	0.7117	0.81012	0
w C-CRNE	0.6979	1.0	0.8221	0.43286	0
PUFS	0.9841	0.9999	0.9919	0.00821	9.14e−6

(b) C–CRNE. It is a clustering-based method for Collecting Reliable Negative Examples. It clusters all the samples into many clusters and takes those without any positive samples as the RN.

The result shows that the precision of using K-Means and C–CRNE is much lower than using the Spy method. We observe that the RN samples extracted by them are far less than those by the Spy method (see Table 4).

4.3.2. Comparison among feature selection methods

To examine the performance of the feature selection scheme based on the reinforcement learning we designed, we also implement several commonly used feature selection methods to make comparison as follows.

(a) K-Best (Giarelis et al., 2020). It ranks features by their χ^2 scores with the label vector, and selects the K features with highest scores.
(b) LASSO (Muthukrishnan and Rohini, 2016). It drops the feature that the coefficient is shrunk to 0 by using $L1$ penalty.
(c) Decision Tree Recursive Feature Elimination (DT-RFE) (Thakkar and Lohiya, 2021). It trains a classifier by all features and scores the importance of each feature by the classifier. The least important features are deselected and a new classifier will be trained by the remaining features, then it processes recursively until the desired number of features is reached.
(d) PCA (Ikram and Cherukuri, 2016). It firstly calculates the covariance matrix and selects the principal components using PCA. Then it calculates the contribution of each feature to these principal components and sorts them according to the contribution value.

Among them, K-Best is one of the filter methods; LASSO and Decision Tree Recursive Feature Elimination (DT-RFE) belong to embedded methods; and AutoFS is a wrapper method. To make a more comprehensive comparison, we also test a classic unsupervised feature selection method based on PCA. We compare the F1-Score on the Ground Truth test between PUFS and PUFS with (w) these methods, as shown in Table 5.

In this test, we also provide parameter τ to limit the feature extraction time of these methods' results. In particular, for methods that can set the parameter N to specify the number of features of the selected feature subset, we set N to meet the maximum number of features that can be accommodated by the feature subset whose extraction time is less than or equal to τ. For the method that searching on the whole feature subset space (AutoFS), we cannot limit the number of feature subsets in this way without improvement, so we test it without limiting its time for feature extraction.

The performance of PUFS on Ground Truth is significantly better than that of using other non-wrapper feature selection methods, as the FPR (0.00830 with AutoFS and 0.00821 of PUFS) is six times less than their. The features selected by the unsupervised method (PCA) are conducive to the detection of positive samples, but it will cause a large number of false positives in the detection model as we can see. AutoFS achieves a similar high F1-Score. But as we aforementioned, its time of feature extraction cannot be limited. In this test, its selected feature subset need at least 24 machines to extract features.

Table 5
Overall performance of feature selection algorithms.

	Precision	Recall	F1-Score	FPR	FNR
w K-Best	0.9545	0.9999	0.9767	0.04773	3.05e−6
w LASSO	0.9491	0.9998	0.9738	0.05358	2.54e−4
w DT-RFE	0.8926	0.9999	0.9397	0.13812	4.26e−6
w PCA	0.6972	1.0	0.8216	0.43418	0.00
w AutoFS	0.9839	0.9999	0.9918	0.00830	9.14e−6
PUFS	0.9841	0.9999	0.9919	0.00821	9.14e−6

Table 6
Overall performance of PUFS using different classifiers.

	Precision	Recall	F1-Score	FPR	FNR
w SVM	0.9791	0.9999	0.9894	0.02139	8.53e−5
w XGBoost	0.9810	0.9999	0.9904	0.01938	6.09e−6
PUFS	0.9841	0.9999	0.9919	0.00821	9.14e−6

4.3.3. Comparison of different classifiers

In the classifier training section (Section 3.3.2), we have declared that the basic classifier is can be replaced by alternative methods. In this section, we test two supervised machine learning algorithms: SVM and XGBoost, and show the performance of PUFS with (w) these classifiers on the Ground Truth test in Table 6. Experiments show that both with SVM and with XGBoost, PUFS can achieve similar results to that with RF.

4.4. Ablation study

In this section, we evaluate the effectiveness of feature selection and PU-learning to verify the contribution of these two parts to our model.

4.4.1. Disable RN extraction

RN extraction is a crucial step of PUFS because our PU learning method will not work if it is disabled. Our strategy is based on two-stage strategies which seeking reliable negative samples from unlabeled samples by RN extraction step. We believe that the classifier trained on positive samples and such reliable negative samples can effectively reduce the impact of positive samples which are in the unlabeled dataset.

To examine the function of this step, we design the ablation experiment of RN extraction, as follows:

- Remove RN extraction from PUFS. When this happens, the data input to the feature selection part and classifier training part will only contain positive and unlabeled samples.
- Label unlabeled samples as negative samples. If the classifier does not receive the RN dataset, its iterative process will fail and work as a supervised binary classifier training as the original. The feature selection part will use the F1-Score of the classifier to calculate the reward value.

We show the experimental result of PUFS without RN extraction in Fig. 9.

We observe that disabling RN extraction has little impact on the performance of PUFS in the Ground Truth test and the feature extraction time after feature selection. Still, its FNR changes back to be very high in the LOGO test. This means the model has a weak ability to detect unknown mAGDs, just like the supervised method (see Fig. 8).

4.4.2. Disable feature selection

Feature selection effectively shortens the time of feature extraction as well as the machines "consumption" and ensuring the detection performance of the model. When feature selection is removed, the model will use all features for training and

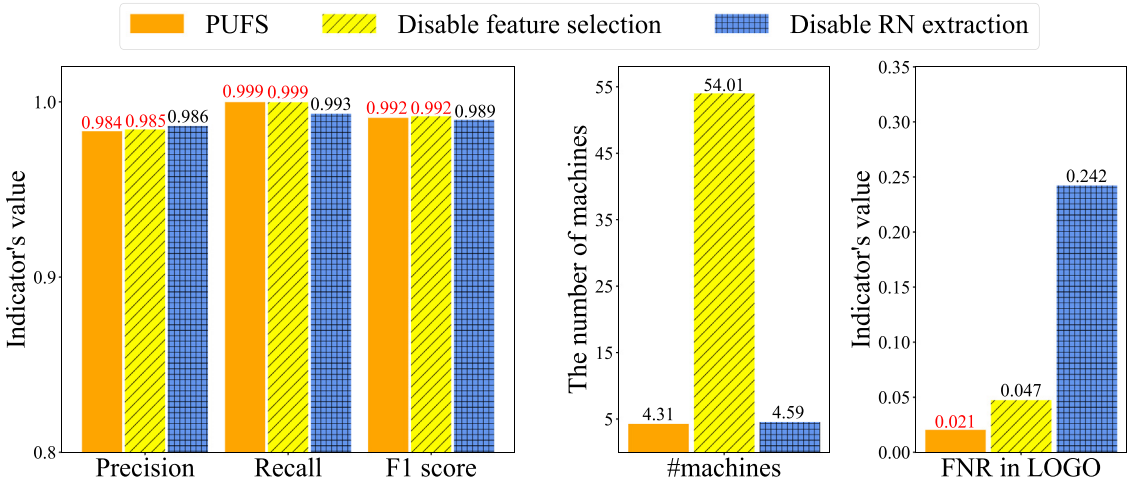
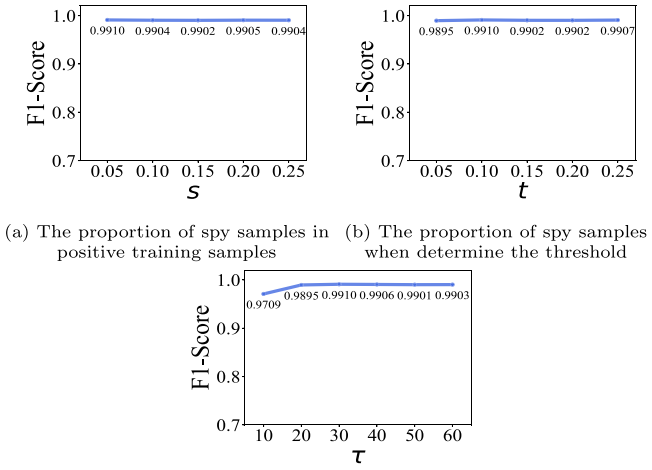


Fig. 9. Overall performance of PUFS, PUFS without RN extraction and PUFS without feature selection.



(a) The proportion of spy samples in positive training samples

(b) The proportion of spy samples when determine the threshold

(c) The constraint of time

Fig. 10. The F1-Score of PUFS as the parameters vary.

detecting. To verify the effectiveness of the feature selection, we directly put the all-features data into the two-stage strategy for PU-learning: RN extraction and classifier training, to compare the detection performance and feature extraction time of PUFS with and without feature selection.

As Fig. 9 shows, when the feature selection part is disabled, the number of "consuming" machines will be very large, which increases machine consumption too much for online real-time detection of a large enterprise.

4.5. Parameter determination

In this section, we evaluate the effectiveness of the hyper parameters in PUFS, and discuss how we determine their value.

We increase s from 5% to 25%, t from 5% to 25% and τ from 10 to 60. As shown in Fig. 10, PUFS achieves very approximate F1-Score when s and t changes, which means these hyper parameters in Spy Algorithm do not impact the final performance of PUFS. We set these parameters as our model achieves the best F1-Score, i.e., $s = 5\%$, $t = 5\%$. For the parameter τ, we find that the performance of PUFS is the best when set to 30.

4.6. The result of the optimal selected features

The set of selected features of PUFS may be slightly different when repeating the experiments, although they all eventually have similar performance. Under the setting of $\tau = 30$, the number of features in the optimal feature subset is around 10. In many experiments, we have observed that several features appear in most selected feature subsets. Specifically, the most frequent structural features are the domain name length, the number of subdomains, the ratio of digital exclusive subdomains, and the ratio of hexadecimal parts. The ratio of digits, the alphabet size, and the digital letter count are the most frequent to be chosen in the linguistic features. In statistical features, the gibberish score and the 25% percentile of 1-gram are always be selected.

5. Related work

A large number of works have been conducted to detect malicious domain names generated by DGAs in the literature (Zhauniarovich et al., 2018; Bilge et al., 2011; Antonakakis et al., 2012; Schiavoni et al., 2014; Schüppen et al., 2018; Wang et al., 2017; Truong and Cheng, 2016; Tong et al., 2019). EXPOSURE (Bilge et al., 2011) extracted 15 network traffic features to discover malicious anomalous domain names and used a decision tree to train a classifier. These features include four traffic duration features, four DNS response features, five time-to-live (TTL) features, and two domain name features. Pleidas (Antonakakis et al., 2012) leveraged a combination of clustering and classification methods. It clustered domains relying on the make-ups of domain names and the groups of the hosts that queried these domains. Phoenix (Schiavoni et al., 2014) carried out mAGD detection based on not only the features of domain names but also those of IP addresses. It applied the DBSCAN clustering strategy to cluster samples. Wang et al. (2017) proposed a DBod scheme to detect mAGDs, which was based on the unsupervised algorithm, Chinese Whispers, for clustering samples. They also applied the query behavior of hosts as one of the features for model training. All of the four methods detected mAGDs according to the features of domain names and network traffic-related features, namely, traffic duration, DNS response, TTL, query behaviors of hosts, and IP addresses. Network traffic data is so confidential and important for dependable systems such as online banks and online shopping services that network operators barely share this data. Therefore, we can only obtain domain names, and the above methods are

inappropriate in our scenario. Additionally, because the traffic data in today's large-scale dependable systems are pretty huge, conducting mAGD detection based on so many types of features is computationally inefficient.

In addition to the above methods relying on network traffic features, a collection of mAGD detection methods based on the features of domain names only have been proposed. For example, Truong and Cheng (2016) applied two features of domain names, i.e., domain names' length and expected values, and chose the J48 decision tree algorithm for binary classification. They used the domain names collected from Alexa as benign existent domain names and treated others as mAGDs. However, a non-existent domain name does not necessarily represent a malicious domain name. There are many benign non-existent domains (bNXDs) that spelling errors and misconfigurations could produce. In this work, our objective is to distinguish mAGDs from bNXDs, and thus this method is inappropriate in our scenario. D3N (Tong et al., 2019) applied convolutional neural networks (CNN) to identify mAGDs from NXDomains. To train an accurate mAGD detection model, it used the mAGDs collected from DGArchive as positive samples. Moreover, it applied the non-existent domain names obtained from a campus DNS resolver as bNXDs. Similarly, FANCI (Schüppen et al., 2018), a state-of-the-art domain name-based mAGD detection method published in USENIX Security, also leveraged the mAGDs collected from DGArchive as positive samples and considered other non-existent domain names as bNXDs. However, both D3N and FANCI ignored the mAGDs that DGArchive cannot cover. Thus, neither can be applied to the partially labeling scenario where only part of mAGDs are labeled because new types of mAGDs are continuously generated.

6. Conclusion

In this work, we present PUFS, integrating PU learning and feature selection, to achieve accurate training based on partial labels and efficient online detection for large-scale dependable systems. It applies a novel, simple, yet effective three-step strategy, including RN extraction, multi-objective reinforcement learning-based feature selection, and classifier training to address the challenge in combining PU learning and feature selection. We believe that the novel three-step strategy can be applied to far more scenarios beyond mAGD detection. We conduct extensive evaluation experiments through the domain name data collected from a top-tier global online bank, demonstrating that PUFS achieves 99.19% of F1-Score, and improves the feature extraction efficiency by 11.53 times compared to a state-of-the-art method. In the future, we will apply more data from other large-scale dependable systems to demonstrate PUFS's performance.

Declaration of competing 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.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. We thank Baojun Liu for his helpful discussions on this work. The work was supported by the National Key R&D Program of China (Grant No. 2021YFB0300104), the National Natural Science Foundation of China (Grant No. 61902200 and 62072264), the China Postdoctoral Science Foundation (2019M651015), and Beijing National Research Center for Information Science and Technology (BNRist).

References

Ahluwalia, A., Traore, I., Ganame, K., Agarwal, N., 2017. Detecting broad length algorithmically generated domains. In: International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. Springer, pp. 19–34.

Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D., 2012. From throw-away traffic to bots: detecting the rise of DGA-based malware. In: Presented as Part of the 21st {USENIX} Security Symposium ({USENIX} Security 12). pp. 491–506.

Bekker, J., Davis, J., 2020. Learning from positive and unlabeled data: A survey. Mach. Learn. 109 (4), 719–760.

Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M., 2011. Exposure: Finding malicious domains using passive DNS analysis. In: Ndss. pp. 1–17.

Bruna, J., Zaremba, W., Szlam, A., LeCun, Y., 2013. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.

Chandrashekar, G., Sahin, F., 2014. A survey on feature selection methods. Comput. Electr. Eng. 40 (1), 16–28.

Chaudhari, S., Shevade, S., 2012. Learning from positive and unlabelled examples using maximum margin clustering. In: International Conference on Neural Information Processing. Springer, pp. 465–473.

Chen, Y., Yan, S., Pang, T., Chen, R., 2018. Detection of DGA domains based on support vector machine. In: 2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC). IEEE, pp. 1–4.

Deng, X., Li, Y., Weng, J., Zhang, J., 2019. Feature selection for text classification: A review. Multimedia Tools Appl. 78 (3).

2022. Domain generation algorithm. URL https://en.wikipedia.org/wiki/Domain_generation_algorithm.

Fan, W., Liu, K., Liu, H., Wang, P., Ge, Y., Fu, Y., 2020. AutoFS: Automated feature selection via diversity-aware interactive reinforcement learning. In: 2020 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 1008–1013.

Giarelis, N., Kanakaris, N., Karacapilidis, N., 2020. An innovative graph-based approach to advance feature selection from multiple textual documents. In: IFIP International Conference on Artificial Intelligence Applications and Innovations. Springer, pp. 96–106.

Ikram, S.T., Cherukuri, A.K., 2016. Improving accuracy of intrusion detection model using PCA and optimized SVM. J. Comput. Inf. Technol. 24 (2), 133–148.

Khormali, A., Park, J., Alasmary, H., Anwar, A., Saad, M., Mohaisen, D., 2020. Domain name system security and privacy: A contemporary survey. Comput. Netw. 107699.

Lanzi, P.L., 2000. Learning Classifier Systems: From Foundations to Applications, no. 1813. Springer Science & Business Media.

Li, G., 2013. A survey on postive and unlabelled learning. In: Computing and Information Sciences. University of Delaware.

Liu, B., Dai, Y., Li, X., Lee, W.S., Yu, P.S., 2003. Building text classifiers using positive and unlabeled examples. In: Third IEEE International Conference on Data Mining. IEEE, pp. 179–186.

Liu, K., Fu, Y., Wu, L., Li, X., Aggarwal, C., Xiong, H., 2021. Automated feature selection: A reinforcement learning perspective. IEEE Trans. Knowl. Data Eng..

Liu, L., Peng, T., 2014. Clustering-based method for positive and unlabeled text categorization enhanced by improved TFIDF. J. Inf. Sci. Eng. 30 (5), 1463–1481.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Muthukrishnan, R., Rohini, R., 2016. LASSO: A feature selection technique in predictive modeling for machine learning. In: 2016 IEEE International Conference on Advances in Computer Applications (ICACA). IEEE, pp. 18–20.

Plohmann, D., Yakdan, K., Klatt, M., Bader, J., Gerhards-Padilla, E., 2016. A comprehensive measurement study of domain generating malware. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 263–278.

Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S., 2014. Phoenix: DGA-based botnet tracking and intelligence. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 192–211.

Schüppen, S., Teubert, D., Herrmann, P., Meyer, U., 2018. {FANCI}: FEature-based automated nxdomain classification and intelligence. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1165–1181.

Song, F., Guo, Z., Mei, D., 2010. Feature selection using principal component analysis. In: 2010 International Conference on System Science, Engineering Design and Manufacturing Informatization, Vol. 1. IEEE, pp. 27–30.

Tang, R., Huang, C., Zhou, Y., Wu, H., Lu, X., Sun, Y., Li, Q., Li, J., Huang, W., Sun, S., et al., 2020. A practical machine learning-based framework to detect DNS covert communication in enterprises. In: International Conference on Security and Privacy in Communication Systems. Springer, pp. 1–21.

Thakkar, A., Lohiya, R., 2021. Attack classification using feature selection techniques: a comparative study. J. Ambient Intell. Humaniz. Comput. 12 (1), 1249–1266.

Tong, M., Sun, X., Yang, J., Zhang, H., Zhu, S., Liu, X., Liu, H., 2019. D3N: DGA detection with deep-learning through NXDomain. In: International Conference on Knowledge Science, Engineering and Management. Springer, pp. 464–471.
Truong, D.-T., Cheng, G., 2016. Detecting domain-flux botnet based on DNS traffic features in managed network. Secur. Commun. Netw. 9 (14), 2338–2347.
Wang, T.-S., Lin, H.-T., Cheng, W.-T., Chen, C.-Y., 2017. DBod: Clustering and detecting DGA-based botnets using DNS traffic analysis. Comput. Secur. 64, 1–15.
Zhauniarovich, Y., Khalil, I., Yu, T., Dacier, M., 2018. A survey on malicious domains detection through DNS data analysis. ACM Comput. Surv. 51 (4), 1–36.

Yongqian Sun received the B.S. degree instatistical speciality from Northwestern Polytechnical University, Xi'an, China, in 2012, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2018. He is currently an Assistant Professor at the College of Software, Nankai University, Tianjin, China. His research interests include anomaly detection, root cause localization in large-scale software systems.

Kunlin Jian is a master student in the College of Software, Nankai University. His research interest focuses on anomaly detection.

Liyue Cui is a master student in the College of Software, Nankai University. Her research interest focuses on anomaly detection.

Guifei Jiang received the B.S. and M.S. degrees from the School of Computer Science and Technology, Southwest University, Chongqing, China, in 2010 and 2013, respectively, and the Ph.D. degree in computer science from Western Sydney University, Sydney, Australia, in 2017. She is currently an Assistant Professor at the College of Software, Nankai University, Tianjin, China. Her current interests include knowledge representation and reasoning and multi-agent systems.

Shenglin Zhang (Member, IEEE) received the B.S. degree in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2017. He is currently an Associate Professor with the College of Software, Nankai University, Tianjin, China. His current research interests include failure detection, diagnosis, and prediction in large-scale software systems.

Yuzhi Zhang received the B.S. and M.S. degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, in 1985 and 1987, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 1991. He is currently the Dean of the College of Software, Nankai University, and a distinguished professor. His research interests include deep learning and other aspects of artificial intelligence.

Dan Pei (Senior Member, IEEE) received the B.E. and M.S. degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA), in 2005. He is currently an Associate Professor with the Department of Computer Science and Technology, Tsinghua University. His research interest includes service management in general. He is a Senior Member of the ACM.