Analyzing the robustness of PUA behavioral machine learning-based detection models

No Author Given

No Institute Given

Abstract. Many applications, and utilities , are distributed freely via popular download sites in an attempt to increase the application's user base. When such applications also include functionalities which are added as a means of monetizing the applications and may cause inconvenience to the user or compromise the user's privacy, they are referred to as potentially unwanted applications (PUAs). Commonly used methods for detecting malicious software cannot be applied to detect PUAs, since they have a high degree of similarity to benign applications and require user interaction for installation. Previous research aimed at detecting PUAs has relied mainly on the use of a sandbox to monitor the behavior of installed applications, however, the methods suggested had limited accuracy. In this study, we present a machine learning-based approach for detecting PUAs, which can be applied on the target endpoint directly and thus can provide protection against PUAs in real-time, and is resilient to commonly used adversarial methods. Our evaluation of the proposed method on a dataset of 1,370 applications (out of which 675 are PUAs), demonstrates its ability to detect 98% of the PUAs with zero false positive rate.

Keywords: Potentially unwanted applications, Machine learning, Antivirus, Adversarial machine learning, Robustness evaluation.

1 Introduction

Nowadays, many applications are available for free, via popular download sites. Application distributors employ tactics in order to increase their user base. Examples of such applications include antivirus software, download managers, backup utilities, dictionaries, Web browsers, and other utilities. While these applications are provided free of charge and thus are attractive to many users, there is a cost to the distributors who provide the applications as they must find ways of making a profit, while covering the costs associated with the free applications (e.g., fees for back end servers).

Potentially unwanted applications (PUAs) are commonly used to serve as a source of income for distributors of free applications. In addition to their core functionality, PUAs often include functionalities that are difficult to remove and may cause inconvenience to the user (e.g., slow down the computer, compromise the user's privacy, or lure the user into paying for unnecessary services). Such functionalities include advertising, user profiling, changing default settings (e.g., to promote software and services), modifying the system configuration [13], and performing security modifications that can create a security threat [1].

Similar to malware, PUAs also attempt to avoid antivirus software and other detection mechanisms (e.g., Google's Safe Browsing API detection [13]). However, while there has been significant attention on malware detection over the years, there has been little focus on PUAs among the research community. Geniola *et al.* [1] created a cross-platform detection framework to detect PUAs by launching them in an automated analysis sandbox. Their framework included the simulation of operating system events, in order to click through the PUAs' displayed offers, or user consent forms which are required in order to install PUAs. However, their solution had several limitations: 1) it takes several minutes for each application to be analyzed, and it cannot be applied in real-time, 2) their UI engine could only automate 67% of the tested applications, and 3) in some cases, the proposed solution failed, since the OS event simulation was unsuccessful for some PUAs (e.g., when a simple change to the user interface is applied). In this paper, we propose a practical approach for the accurate detection of PUAs.

The proposed approach can be applied on the endpoints, without the need for external testing in a sandbox environment. Our solution extracts several *bundle installation* features during application installation, including 'running as administrator, 'number of processes created, 'number of DNS requests', 'number of folders created', and applies a machine learning approach, in order to classify the file as a benign application or PUA. Our method can be tuned in such a way that PUA distributors (also known as affiliate networks) will be forced to install fewer applications per installation in order to evade the detection model, thereby reducing their profit and motivation. To evaluate our proposed method, we tested 1,370 applications (out of which 675 were PUAs) from 40 different websites listed on Appendix I, and monitored their behavior during installation on a Windows 7.0 OS endpoint, and windows 10.0 OS endpoint, on which antiviruses were installed. VirusTotal was used for classifying the applications as benign or PUA. If at least one of VirusTotal's significant engines (e.g., Avast, Symantec) detected the file as a PUA, or that our manual testings found behavior, which was not authorized by the user (advertisement, additional applications that user did not agree to install, decrease system performance, or user's experience) it was labeled as 'PUA,' otherwise it was labeled as 'benign.' We have also implemented a C++ Windows application, which can monitor any running application, to test in real-time if it is a PUA or benign application.

The results show that the rotation forest [10] classifier was capable of detecting 98% of the PUAs with zero false positives, with an AUC of 0.99. Based on these results, we assume, that the proposed method can be used to efficiently differentiate a PUA from benign applications during installation, and thus can prevent PUA installation by terminating related processes. Furthermore, because of the features that are used for classification, an attempt to evade the proposed solution will result in a significant reduction in the PUA distributor's profit, which is already low [19], since each successfully installed application increases the profit. To summarize, the contributions of this paper are as follows:

1. We propose a machine learning-based method for detecting PUAs, which can be simple enough to implement, yet it is accurate, and resilient to commonly used adversarial methods. We implement and evaluate our proposed method on a dataset of 675 PUAs and 675 benign samples, and we have made this dataset publicly available to the research community.
2. Unlike previous studies that focused on sandbox-based detection, we focus on real-time detection. We implement an application for detecting PUAs during their installation. When using our application, the installation can be blocked when a PUA is detected, by terminating related processes.
3. We test our model, using AV evasion techniques, and adversarial machine learning methods, including GAN, and using gradient based attacks, to create samples and modify PUAs, so they will be able to bypass the model, and show that even using such methods, it is not profitable to distribute PUAs, once our method is in use.

The rest of this paper is structured as follows. Sections 2 and 3 discuss the background and related work. Section 4 presents our detection methodology. Section 5 describes the experimental evaluation. Section 6 discusses some adversarial machine learning methods to attack the model and show their impact on PUA's profit, and Section 7 discusses the conclusions. A preliminary version of this research was previously published as a short paper, but the current paper has more than 50% of new material, including, but not only extended information on PUAs' economy, robustness evaluation, additional features, cross platform PUAs' features comparison, testing on multiple Windows' OS versions, and a significantly larger dataset.

2 Background: Pay Per Install (PPI)

The key players in the PPI business model include (illustrated in Figure 1):
Application advertiser (1). A person or company that wants to distribute his/her application (e.g., [13]) to a large group of users. Such an application is referred to as the advertiser's component. Note, that while some of the applications distributed are benign (e.g., Opera), others are not and may contain unwanted functionalities such as advertisements (e.g., shoppers) [13]. The requested software, which the user was searching for, is called a carrier, because it 'carries' the components (advertisers' applications) of multiple advertisers, who pay a fee for their distribution.
PPI network (2). PPI networks (e.g., www.payperinstall.com) are the mediators between application advertisers

and publishers. PPI networks create the bundle installers which contain several applications from different application advertisers. A bundle installer contains one *carrier application*, which is the main legitimate application that a user will search for and want to install (e.g., Opera). It also contains applications that the user may not be interested in installing. The bundle installers are created based on features extracted from the target machine and user profile, such as geographical location, operating system, or antivirus (AV) installed (extracted in order to enable the bundle installer to evade detection). For example, in the US, applications like Yahoo Toolbar are more likely to be installed, since users in the US are more likely to use Yahoo and are more familiar with its portal. In contrast, in less economically developed countries a common application may be a free AV, like Baidu. PPI networks also provide monitoring and installation statistics based on which the payments to the publisher are calculated and made.

Affiliates (3). Affiliates (also known as publishers) are software application distributors. Affiliates usually conduct marketing campaigns and place advertisements in an attempt to attract potential users and lure them into downloading the free applications advertised. Affiliates pay to run such marketing campaigns and in turn receive a payment from the application advertiser for each successful installation reported.

End user (4). A person who tries to download and install a free application (e.g., Windows OS, as seen in Figure 1).

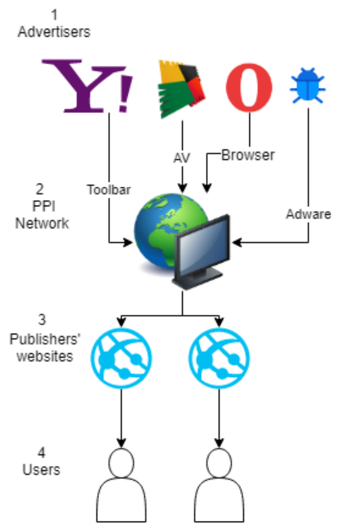

Fig. 1: Key players in the PPI business model.

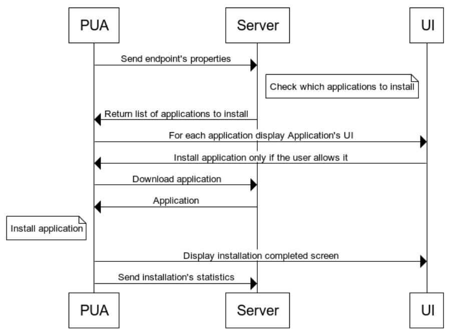
PUA
Server
UI
Send endpoint's properties
Check which applications to install
Return list of applications to install
For each application display Application's UI
Install application only if the user allows it
Download application
Application
Install application
Display installation completed screen
Send installation's statistics
PUA
Server
UI

Fig. 2: PUA installation sequence diagram.

Profit is the motivating factor for the affiliates and PPI networks that spread both benign and unwanted software. A typical PUA installation scenario will start with an end user that wants to download a free software application (see Figure 1 (4)), like the Opera Web browser.

Usually, the user searches for this application in a search engine, clicks on a sponsored advertisement, which directs him/her to a publisher's website (Figure 1 (3)), and downloads what he/she believes to be the requested software, but is in fact a PUA (Figure 1 (2)). The download site, often called a landing page, is prepared by the publisher, who pays to advertise his/her landing page, so users will reach it when they search for the Opera Web browser or another application. The affiliate network is the entity that creates the bundle installer and monitors the installation statistics. Upon successful installation, the advertiser makes a payment to the affiliate network, however most of the payment is directed to the publisher who assumes the greatest risk in this scenario, since the publisher must pay for the Web advertisements in advance. The remaining profit goes to the affiliate network. The installation sequence of the PUA, is listed in Figure 2.

2.1 PUA's Economy

The pay per install model motivates application distributors to bundle several third party applications with their own application in return for a fee, for each installed application. The payment to the PPI network by the advertisers is sometimes made up front, in return for a fixed number of installations. Affiliate networks profit from installation fees paid by the advertisers. Install rates usually vary from \$0.1-\$0.18 in the most demanded regions (like US or UK), to \$0.07-\$0.08 in the least popular ones (like most of Asia). In order, to avoid detection by AV, affiliate networks use packers, which are developed by 3rd party companies, change the program size, and add detection of virtual machines, and debuggers, which are commonly used by analysts [9]. Some of the PUAs, use command and control protocols, to get a list of specific applications to download, according to the user's location, installed operating system, and installed AV (see Figure 2). It is also common, that affiliate networks, also act as publishers, and install other affiliate networks' installers, during their installation [9]. We assume that this is done due to the low profitability of this industry, which forces affiliate networks to try to maximize the profit from each installation. Based on Google Safe Browsing telemetry [13], in 2016, PPI networks resulted in over 60 million download attempts each week (nearly three times that of malware). While antivirus programs and browsers try to protect users from unwanted software, PPI networks actively interfere with or evade detection [13]. According to the 2019 Quick Heal Threat Report,[1] in the second quarter of 2019 there were 19 million PUA installations, so it seems, this threat (PUA) is still relevant. The PPI model described above is used in Section 4, when we discuss our PUA detection methodology.

[1] `https://www.quickheal.co.in/documents/threat-report/qh_threat_report_q2_2019.pdf`

3 Related Work

In the most relevant paper on PUA detection [18], the author presented several features that can be used with a machine learning PUA classifier, with the aim of creating a set of detailed guidelines to help define PUAs in today's marketplace and inform users about potentially risky applications. Previous work on PUA detection focused primarily on the PPI business model [13] and PUA distribution via download portals analyzed by Rivera *et al.* [20]. Thomas *et al.* [13] presented a comprehensive survey of PUA distribution and discussed its economic aspects.The authors demonstrated the deceptive methods used by PPI networks to avoid detection. Kotzias *et al.* [19] also discussed the economic aspects of PUAs. The authors analyzed several commercial PPI services and addressed issues such as the profitability of commercial PPI services and the operations running them, and the revenue sources of the operations. Their main goal was to evaluate the economics of commercial PPI services used to distribute PUAs. based on their assumption that understanding their economic evolution over time is essential for evaluating the effect of deployed defenses. In this paper we also consider the economics of PUAs and their ecosystem and demonstrate how our solution can decrease a PUA's profit (see Section 4). Several papers discussed the detection of PUAs and the challenge of distinguishing them from legitimate applications. Kwon *et al.* [3] presented a system for detecting silent delivery campaigns for the distribution of malware which do not require user interaction. Such a system is not effective for PUA detection, since a PUA requires user consent for each application it installs. Geniola *et al.* [1] proposed a sandbox-based architecture for detecting PUAs. Their analysis was based on almost 800 installers, downloaded from eight popular software download portals.In their architecture, the authors used an agent that tries to detect when an installer is waiting for user input and then sends the input event to the installer, which is most likely to advance the installation process. Their solution required the use of a sandbox environment and took a few minutes to analyze each application. Moreover, the proposed solution is not robust, since a PUA's user interface must be known, in order to correctly simulate PUA installation by a user. Stavova *et al.* [24], conducted a survey of AV beta users to gauge their interest in deploying a PUA detection mechanism based on several warning messages options. Their results indicated that 74.5% of the users would choose to use such a detection mechanism. In a followup paper, the authors conducted another large-scale experiment, in which they used different warning messages designed to encourage users to enable the PUA detection mechanism when installing a security software solution from the Internet [24]. Another study discussed the detection of PUAs in a mobile environment [23], mentioning key PUA indicators, such as functionality to root or jailbreak a device, remote monitoring, cracked or repackaged applications, and excessive advertisement packages. Since in our research we focus on desktop computers and the Windows environment, most of the features identified by [23] are not relevant for Windows PUAs.
In another study, Hatada *et al.* [8], they

developed a system called CLAP that detects and classifies PUAs. Their approach leverages DNS queries made by mobile applications. Using a large sample of Android apps from third-party marketplaces, they first reveal that DNS queries can provide useful information for detection and classification of PUAs. Afterwards, they have validated that existing DNS blacklists are limited when performing these tasks and have clarified that the CLAP system performed with high accuracy. The idea was to remove common fully qualified domains names (FQDNs), and find PUAs by their access to advertising domains (e.g. mob.guohead.com). This approach can result in false positives, since using unfamiliar domains, does not necessarily means there is a malicious/unwanted behavior. In another article, Maurice *et al.* [16], researchers automatically downloaded and installed application from 5 most popular download sites, using an automated process, which clicks through the installation, looking for "finish", and "next" buttons, and used AV scanning to classify the installers to different categories, however, it is not related to PUA detection, except that, one of the categories, was if AV marked application as PUA. In our study, we use one of the features proposed in [18] (see Section 4).

Related Work On Adversarial Learning Machine learning models are known to lack robustness against inputs crafted by an adversary. Such adversarial examples can, for instance, be derived from regular inputs, by introducing minor yet carefully selected perturbations [12]. Furthermore, there is another method, that adversaries can try to use to bypass detection, which is code obfuscation techniques for malware detection evasion, suggested by Rozenberg *et al.* [21], which include: modifying the system call parameters; adding no-ops, which means system calls without any effect; and developing equivalent attacks, choosing an alternative system call sequence which will result in the same effect. This will not create a problem, for the proposed solution, since our application monitors file system changes, the created processes, DNS requests, process privileges, and does not inspect specific system calls or API usage.

PUA Detection vs. Malware Detection. In the past decade, there has been significant investment by the research community in developing novel techniques for malware detection. For example, Ding *et al.* [27] presented a deep learning-based method for automatic malware signature generation and classification. The proposed method was based on API calls and their parameters, registry entries used, websites accessed, and ports. Another malware detection method proposed, analyzed the entire executable file [5]. Such malware detection methods are incapable of detecting PUAs, since they don't consider whether the EXE requires admin permissions in order to run, how many DNS requests it sends, how many folders the EXE creates on the file system, and user interaction, which is critical for a PUA in order to obtain user consent. As we will show later in the paper, such features are very useful for PUA detection (see Section 5). Moreover, such malware detection methods are based on the API, registry, file system, or port usage. In addition, many automated malware analysis sandboxes can be detected by taking advantage of the artifacts that affect their virtualization engine [23].

4 The Methodology

We present a unique, yet practical method that allows users or security software to accurately identify PUAs, without the need for a sandbox. The general idea behind our method is that a PUA is basically a dynamically bundled application. This means that it installs several applications which are selected at run-time based on a user's computer characteristics. In order to perform system changes, during runtime, several processes are created, several folders are created on the file system, and the PUA connects to a server and runs as administrator. In order to evaluate our methodology, we have conducted a software installation experiment, in which we evaluated the effectiveness of a machine learning classifier, in order to distinguish between PUAs and benign applications (see Section 5 for details). Specific PUA behavior is not malicious by nature, but it will allow us to make a distinction between a PUA and benign applications. After creating a machine learning classifier, a monitoring application was used to implement the detection and termination of the PUA installation, using this classifier.

Machine learning classifier's suggested features

We propose a run-time, machine learning-based classifier for PUA detection. The proposed method utilizes the following set of features (some of which have been used in previous studies [18]):

1. Uses IE control (uses Internet Explorer Control in an MFC dialog) – a Boolean feature, suggested in [18], which indicates if an application uses Internet explorer Control in its user interface. This feature is included, because some PUAs use IE Control to present dynamic offers to users.
2. Runs as admin – a Boolean feature, suggested in this research, which checks whether an application runs with administrator privileges. This feature is included, because a PUA needs to run with admin privileges, in order to change system settings and access admin protected folders, such as 'program files', where applications are installed.
3. Number of processes created – a feature, suggested in this research, which indicates the number of processes created by the application. This feature is included, because each application is installed as a separate process; a PUA runs multiple processes in order to detach itself from malicious activity.
4. Number of main folders – an integer feature, suggested in this research, which indicates the number of folders created by the application during installation. This feature is included, because each application installed by a PUA is created in a different folder.
5. Downloads additional installers – an integer feature, suggested in this research, which indicates the number of applications downloaded by an application. This feature is included, because some PUAs dynamically download components according to the user configuration and geographic location.

6. Creates/modifies kernel drivers (.SYS files) – a Boolean feature, suggested in this research, which indicates whether an application creates or modifies existing kernel drivers. This feature is included, because some PUAs use kernel drivers in order to inject advertisements into the websites visited.
7. Imported DLL names include 'oleaut32.dll' – a Boolean feature, suggested in this research. This feature is included, because some PUAs use an OLE (object linking and embedding) library in order to enable a component object model (COM) framework.
8. Number of imported DLLs – an integer feature, suggested in this research. This feature is included, because some PUAs use a limited number of imported DLLs in order to avoid AV detection based on the import table.
9. Number of DNS requests – an integer feature, suggested in this research. This feature is included, based on previous research [8], where PUAs were identified by accessing non commonly used domains.

Note, that a total of eight new features, were introduced in this research. A PUA analyzes the endpoint's characteristics, and then it sends this information to the server. Afterwards, it downloads and installs a list of applications. Note that in some cases a PUA executable can also install applications that are embedded in the setup file itself. Figure 2 presents a diagram of a PUA installation sequence.

Monitor. We created and tested a monitor application for Windows OS, which can be used to terminate a PUA based on the classifier's decision. This monitor contains code which initiates the performance of the following tasks: (1) reads EXE signature, (2) checks if EXE requests static or run-time admin permissions, (3) reads EXE Import table, (4) registers shell events (for file system changes notifications) and WMI process creation events to Windows, (5) checks if EXE listens to a port, (6) checks if EXE uses IE browser control, and (7) calls the classifier with the collected features, every k events on feature changes (e.g., number of created processes); if the classifier answers *yes*, terminate EXE, else continue monitoring.

The monitor is a Windows C++ application, which runs in user mode, that the user will drag and drop an executable into. In return, the monitor performs static analysis of the executable; then the monitor runs the executable and performs dynamic analysis, and the file system changes by registering shell events and processes created while it was running on Windows by using WMI (Windows Management Instrumentation) query language notifications. Moreover, the classifier which the monitor is using at real-time, was created by our experiment.

5 Evaluation

5.1 Dataset

Our dataset contains 1,370 files (675 benign samples and 675 PUA samples), downloaded between January 2019 and December 2020, from 40 different websites, including the most popular download sites: filehippo.com, cnet.com, and softonic.com [1]. Among other files, our dataset includes legitimate bundle installers such as Adobe Flash Player (which installs two security products) and Avira AV (which is bundled with Opera Web browsers); these applications were downloaded from the vendors' sites.

In order to label the files as benign or PUA, we scanned the files with VirusTotal. A sample that received one or more positive classifications by significant antivirus engines, or that our manual testings found behavior, which was not authorized by the user (advertisement, additional applications that user did not agree to install, decrease system performance, or user's experience) was labeled as PUA. As can be seen in Figure 3, which presents the detection rates of the different antivirus engines in VirusTotal, no single antivirus engine was able to detect more than 15% of the PUAs. Given that only one antivirus is usually installed on an endpoint, it is likely that in most of the cases, a PUA will not be detected. Therefore, in order to improve the labeling of the files, we manually tested and analyzed each application to validate the files' labels.

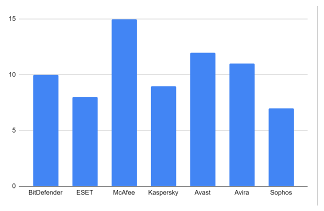

Fig. 3: VirusTotal AV's detection rates.

To perform the analysis, each application was downloaded using Chrome browser on Windows 7.0 OS and Windows 10 OS endpoints installed with

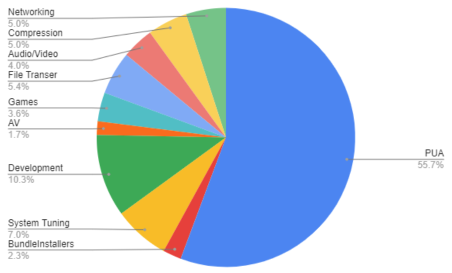

Fig. 4: Categories of installed applications.

Symantec and Windows Defender antiviruses. The downloaded files were checked by the antiviruses using the Safe Browsing API. In addition, we executed each sample on both host machines and monitored it using process explorer, Wireshark (to observe the DNS requests [25], and our own monitoring application.

The downloaded applications included the categories shown in Figure 4 which shows the frequency of each category.

5.2 Proposed Method's Performance

We evaluated our approach with Rotation Forest (produced an AUC of 0.99), XGBoost [11] (produced an AUC of 0.98), Naive Bayes (produced an AUC of 0.98) , and Decision Tree (produced an AUC of 0.98) classifiers. Out of which, the rotation forest produced the most accurate results. The averaged results based on 10-fold cross validations experiment shows that the classifier, when set to produce zero false positives, the classifier detected 98% of the PUAs.

Detection errors (false positives) can occur when an application installs multiple applications (e.g. Adobe PDF reader, also installs McAfee Safe Connect, and Adobe Chrome extension). In addition, such errors can happen when an application creates multiple processes during its installation, creates multiple processes on the file system, runs as administrator, and also accesses multiple network domains. Such a combination was not a frequent one, according to our findings. Our Monitor application, uses the rotation forest classifier to detect if an application is PUA or benign.

5.3 Malware Detection Classifier Performance

In order to verify, that malware classifiers don't effectively detect PUAs, we have extracted malware features [15], from our dataset, and used these features as input for the same classifiers used for PUA detection. The results are depicted in table 1. In addition, we have manually tested 50 PUAs in Crowd Strike falcon sandbox [4], and all of them were marked as clean.

Table 1: Classifiers' Results

Classifier	FPR (TPR=1)	TPR (FPR=0)	AUC
Rotation Forest (PUA Features)	0.002	0.98	0.99
Rotation Forest (MW Features)	0.98	0.01	0.64
XGBoost (PUA Features)	0.02	0.99	0.98
XGBoost (MW Features)	1.0	0.04	0.49
Decision Tree (PUA Features)	0.003	0.98	0.98
Decision Tree (MW Features)	1.0	0.0	0.5
Naive Bayes (PUA Features)	0.002	0.98	0.98
Naive Bayes (MW Features)	0.98	0.01	0.42

‘

6 Robustness of the Proposed Detector to Adversarial Machine Learning Attacks

6.1 Background

In this section, we evaluate the robustness of the proposed detector to adversarial machine learning attack. These methods include:

1. Direct gradient-based attacks - in this method, the model, and weights must be fully known to the attacker, and he must be able to query it. Gradient based adversarial attacks exploit a very simple idea originating from the concepts involved in back-propagation (an algorithm used to train deep neural networks). Gradient based attacks use this concept to develop a perturbation vector for the input by making a slight modification to the back-propagation algorithm [7].
2. Attacks against models, which report a score - in this case the attacker has no knowledge on the model, but can learn the score. This is a more generic attack. An attacker can directly measure the efficacy of any perturbation, according to the model's score.
3. Binary black box attacks - in which the attacker has no knowledge of the model, but might be able to get a label, if a sample is benign or malicious. The idea is to create a substitute model trained to reproduce outputs observed by probing the target model with corresponding inputs. Then, the substitute model is used for gradient computation in a modified GAN (Generative Adversarial Network.

A class of machine learning frameworks, that for a given training set, learns to generate new data with the same statistics as the training set [6]), to produce evasive malware variant [2].

In the next sections, we will present two different approaches for adversarial samples generation, and analyze the robustness of the model to such manipulations, and the likelihood of these changes to be applied, considering the possible financial impact, that will result from their usage. The first approach, will use gradient-based attack, to create a surrogate model to ours, which will allow us to modify PUA samples' features, to benign samples features, so they will be able to bypass our model. The second approach, will take one of the PUAs, and by generating benign samples using GAN, find a benign sample, which is as close as we can to the PUA's features, so we can modify PUA's feature accordingly, to appear as a benign application. Finally, we will demonstrate practical methods, of modifying PUAs' functionality, so they can avoid detection by our model, yet still perform all the software installations done, before these modifications have occurred.

6.2 Using Gradient-based Attack To Evade Detection

This attack is based on the method described in the article [14]. Attacking scheme consists of the following steps:

1. Building a classifier for PUA
2. Creating an embedding model, to convert categorical data to numerical data
3. Creating and training a neural network to convert PUAs to benign samples
4. Using the neural network to convert PUAs to benign samples
5. Converting features back to categorical to the original features' definition
6. Checking if PUA is blocked by our original model

The gradient method attack of [11], uses a Neural network embedding model which requires that all features would be numeric. We have transformed 5 categorical features into matching numeric features, so we can use them as input to the neural network. Rest of the four features have remained intact. In order to create the model, we have used 950 samples for training dataset, 200 samples for test dataset, and 220 samples for the attack. The attacked model is a rotation forest, with an AUC of 0.98. The embedding model has the following characteristics:

1. Embedding Size - 4, Batch Size - 64, and Epochs - 25
2. Layers -
   - Input Layer - 9 neurons, Drop Layer - 6 neurons with Relu activation, Dropout - parameter of 0.3, Dense layer, embedding - 4 neurons

The embedding model has an AUC of 0.99 The attack is performed by changing the model features. The results are depicted in Table 2, Each row presents a different experiment where the attack parameters are: S - maximum number of features to change, the percentage of samples in which 1 or more features were changed, and the attack success rate, which means what percentage of

PUAs were classified by the modified model as Benign.

Table 2: Attacks Success's Percentages

S	1 Feature Change	2 FS	3+ FS	Success
5	53%	15%	32%	22%
5	16%	22%	62%	22%
5	27%	55%	18%	29%
8	58%	4%	38%	48%
8	30%	8%	62%	49%
8	25%	48%	27%	55%

The table shows, that the category of changed features, is more important, than the amount of modified features. Here are the list of the features changes, done in the attack, and their possible impact (note, that running as administrator feature's value is constant, since PUA needs admin's permissions in order for the installation to succeed):

1. Downloads additional installers. Changing this means, the PUA will not download dynamic offers, but only install static content. It is possible, but means less profit, since PUA will not install applications, which were already installed on the users computers.
2. Uses IE control. Changing this means, the PUA will use/not use an internet explorer control for displaying dynamic offers, which is possible, but requires software development efforts.
3. Creates/modifies Kernel drivers. Changing this means, the PUA will use / not use filters of file system/network drivers, which is possible, but requires software development efforts, since instead of injecting advertisements using network driver, it can also be done using a browsers' extension.
4. Number of DNS requests. Changing this means, the PUA can access less domains, and might need to host some of the installed applications, which can add to the cost of each installation.
5. Number of imported DLLs. Changing this means, the PUA will load some of the DLLs dynamically, instead of using static linkage.
6. Number of main folders created. Changing this means, decreasing the number of folders, so the PUA will be able to install less applications, which means less profit per installation.
7. Number of created processes. Changing this means, decreasing the number of processes, so PUA will be able to install less applications, since each application is installed using a different process.
8. Imported DLL names include 'oleaut32.dll'. Changing this means, the PUA will not use COM, or load COM objects dynamically, which is not hard to implement, but requires software development efforts.

Most commonly modified features by our attack, are:

1. 'Number of created folders' - when the attack is allowed to modify all the features, this feature has been modified in 55 out of 190 PUAs, which is equal to 28.9% of the modified applications.
2. 'Number of created processes' - when the attack is allowed to modify all the features, this feature has been modified in 56 out of 190 PUAs, which is equal to 29.4% of the modified applications.

3. Rest of the 41.7% of the modified PUAs, are still caught after their features have been changed.

To summarize, we see that in 58.3% of the samples, PUAs will have to reduce the number of created folders, and created processes, which means they will install less applications, and this will reduce the potential profit per installation. On the other hand, if PUAs will not change the mentioned features, they will still be detected by our model. Even after these changes, 45% of the samples are detected by our model, so PUAs' profit will be dramatically reduced.

6.3 Using GAN Samples To Avoid Detection

Another way to try to attack the model is to generate samples with sets of features which are close to the real set of features, as described in [26]. We have used TGAN python module [26], in order to learn our dataset. and generated 1000 new samples. Based on these samples, an attacker can understand how to modify its PUA, so it will not be detected by our model. We chose one of the PUAs, and reviewed the generated GAN's Samples, for the closest features for a benign application. The PUA and benign features for the selected sample, are listed in table 3. Note that changing the PUA's features to the benign sample's features, means installing less applications. In our case, two applications, instead of four. This means a significant reduction of PUA's profit per installation. Note that changing PUA's features to the benign sample's features, means installing less applications. In our case, two applications, instead of four. This means a significant reduction on PUA's profit per installation.

Table 3: Samples's Features

Admin	**NumProc**	**Folders**	**DNSReq**	**Result**
True	9	4	4	PUA
True	7	2	1	benign

6.4 Practical Evasion Techniques

In this section, we will demonstrate several practical techniques to try to avoid detection by our model. First, since our model does not monitor the processes' memory, it is possible to bypass its detection, by using techniques of process or code injection, like process hollowing [22], which allows replacing target process memory, with source EXE memory, however, such methods are known and suspicious, and are likely to be detected by most AV products. we have tried using this on one of the PUAs, using the following steps:

1. start a benign process, like notepad in suspend mode
2. read PUA EXE from a file
3. unmap executable section of notepad EXE, and free its memory
4. write PUA PE content into notepad in its virtual memory
5. change notepad entry point to the injected process
6. resume notepad thread

There are existing code samples, which allowed us to do this, however, this injection is caught by windows defender.
Manipulating Features In Order To Avoid Detection There is another way, that an adversary can use

to bypass our detection method, for example, by creating a windows application, which performs the same functionality as his PUA, but using a different way, so it will not be detected by our solution. A PUA can use the following flow:

1. Download each application it wants to install, but wait a few minutes, before each download.
2. Create multiple tasks in the task scheduler.
3. Schedule each task to run in a different time.

In windows, each task runs, as a different process, and will install a single application. This way , the end result is that PUA functionality goes undetected by our algorithm, by splitting the unwanted applications installations to multiple task schedulers' tasks. This solution is not hard to implement, however, delaying, and splitting the installation to multiple tasks, increases the complexity of gathering installation statistics, increases possible installation's errors, and will reduce the actual profit, since advertisers pay only on detailed reported installations, with exact installation information, such as IP, geographic location, time of installation, etc.. . Furthermore, we check the impact of modifying additional PUAs' features, by an adversary, in order to bypass detection by our method, and its possible impact on affiliate network's incentives and revenues. Such features include:

1. Not running as administrator. This is very challenging, since the 'program files' folder requires administrator permission in order to modify/create files.
2. Reducing the number of processes. This is possible, but it means installing less applications, which by default are installed to different folders; this means that each installation will yield less money.
3. Reducing the number of folders. This means installing less applications, which will result in a less profitable installation. Furthermore, if the PUA installs several components in a single folder, advertisers will refuse to be associated with such an affiliate network, since such behavior can cause their EXE to be flagged by an AV which will block their future installations.

In short, avoiding our detection model will reduce the potential profit of the affiliate network, and reduce the affiliate network's incentive for installing the PUA.

7 Discussions and Future Perspectives

Potentially unwanted applications are not viruses, nor do they steal the users' sensitive data directly. They do, however, introduce security risks into the system, decrease the system's efficiency and performance, and disrupt the user experience [18]. Previously suggested methods include using sandbox [1]. Alternatively, trying to use sites like VirusTotal, is similar to using sandbox, since they require uploading the application to their web site, and running it in an external environment for dynamic analysis.Our experiment shows that it is possible to create an effective accurate classifier for PUA detection (and thereby prevent their installation), based on machine learning of specific features, including the features of 'running as administrator', 'number of processes created,' 'number of DNS requests', and 'number of folders created.' Such features distinguish PUAs from benign applications. Moreover, we have showed, that our method is resilient to adversarial machine learning, since the impact of bypassing it, is earning much less money from each installation. The limitations of this approach, are that it may result in possible false positives, misclassifying legitimate bundle applications that install several applications in a single EXE. However, it seems that the users expect no more than one or two such applications per installation, so such cases should be rare. Another issue, is that we are closing the PUA during its installation, so it is possible that it will install several applications by the time we stop it, however, since we know when the PUA was running, the user can return to a safe state, by reverting to a windows restore point, before the PUA has been installed. Moreover, our monitor application is implemented as a user mode application, for research and proof of concept. A more secured solution, that cannot be terminated by monitored PUAs should include a kernel mode driver that is more difficult to bypass or temper with. The proposed method can be used in order to reduce the distribution of PUAs and prevent their installation, since even if statistically a PUA will be able to avoid detection by our method, using adversarial machine learning methods, or other evasion techniques, there will be a financial impact on possible profit per installation.Our plans for future work include applying similar methods on other operating systems (e.g., android, or macOS).

Cross Platform Overview Of PUAs' Detection While this article, focuses on Windows's PUA detection. For completeness and possible future extended research, we add a high level overview of PUAs' features on android's mobile platform [23], and MAC operating system [17], on which PUAs have different features, but ML classifiers, can also be used for identifying them.

OS	**Feature**	**Description**
MAC	Bundle install	Installs multiple apps
MAC	Web browsing	New tab's ads
MAC	Web browsing	Changing the search
MAC	Ads	Displaying web ads
Android	Ads	Display Adults' Ads
Android	Privacy Permissions	User's Accounts
Android	Location Permissions	Location, etc...
Android	UX's Permissions	Boot Completed, etc...

References

1. Alberto, G., Antikainen Markku, Aura Tuomas: A large-scale analysis of download portals and freeware installers. In: Proceedings of Nordic Conference on Secure IT Systems. pp. 209–225. Springer (2017)
2. Anderson, H.S., Kharkar, A., Filar, B., Roth, P.: Evading machine learning malware detection. Black Hat (2017)
3. Bum Jun Kwon, Virinchi Srinivas., Amol Deshpande, Tudor Dumitras: Catching worms, trojan horses and pups: Unsupervised detection of silent delivery campaigns. In: Proceedings of NDSS (2017)

4. Crowd strike's automated malware analysis tool, `https:\\crowdstrike.com`
5. Edward, R., Jon, B., Jared, S., Robert, B., Bryan, C., K, N.C.: Malware detection by eating a whole exe. In: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence (2018)
6. Generative advererial networks, `https://en.wikipedia.org/wiki/Generative_adversarial_network`
7. Haldar, S.: Gradient-based adversarial attacks : An introduction (2020), `https://medium.com/swlh/gradient-based-adversarial-attacks%20%20-an-introduction-526238660dc9`
8. Hatada, M., Mori, T.: Clap: Classification of android puas by similarity of dns queries. IET Information Security **E103-D**(2), 265–275 (2020)
9. J Caballero, C Grier, C Kreibich, V Paxson: Measuring pay-per-install: the commoditization of malware distribution. In: 20th {USENIX} Security Symposium ({USENIX} Security 11). pp. 187–202 (2011)
10. Juan, R., Ludmila, K., Carlos, A.: Rotation forest: A new classifier ensemble method. IEEE transactions on pattern analysis and machine intelligence **28**, 1619–30 (2006)
11. Kantchelian, A., Tygar, J.D., Joseph, A.: Evasion and hardening of tree ensemble classifiers. In: Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48. arXiv:2010.03180, PMLR (2016)
12. Kathrin, G., Nicolas, P., Praveen, M., Michael, B., Patrick, M.: Adversarial examples for malware detection. In: European Symposium on Research in Computer Security. pp. 62–79. Springer (2017)
13. Kurt, T., Elices, C.J.A., Ryan, R., Jean-Michel, P., Cait, P., Marc-André, D., Chris, S., Fabio, T., Ali, T., Marc-Antoine, C., et al.: Investigating commercial pay-per-install and the distribution of unwanted software. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 721–739 (2016)
14. Levy, E., Mathov, Y., Katzir, Z., Shabtai, A., Elovici, Y.: Not all datasets are born equal: On heterogeneous data and adversarial examples. arXiv:2010.03180 (2020)
15. Malware detection using machine learning, `https://github.com/tuff96/Malware-detection%20%20-using-Machine-Learning`
16. Maurice, C., Bilge, L., Stringhini, G., Neves, N.: Detection of intrusions and malware, and vulnerability assessment. Nature Public Health Emergency Collection **12223**, 192–214 (2020)
17. Meskauskas, T.: Mac purifier unwanted application (mac) (2019), `https://www.pcrisk.com/removal-guides/13398-mac-purifier%20%20-unwanted-application-mac`
18. Mo, J.: How to identify pua (2016), `https://www.infosecurityeurope.com/__novadocuments/86438?v=635670694925570000`
19. Platon, K., Juan, C.: An analysis of pay-per-install economics using entity graphs. WEIS (2017)
20. Richard, R., Platon, K., Avinash, S., Juan, C.: Costly freeware: a systematic analysis of abuse in download portals. IET Information Security **13**(1), 27–35 (2019)
21. Rosenberg, I., Ehud Gudes: Bypassing system calls-based intrusion detection systems, concurrency and computation. Practice and Experience 29 (2017)
22. Rouse, M.: Process hollowing (2020), `https://whatis.techtarget.com/definition/process-hollowing`
23. Svajcer, V., McDonald, S.: Classifying puas in the mobile environment. In: Virus Bulletin Conference (2013)
24. Vlasta, S., Lenka, D., Vashek, M., Mike, J., David, S., Martin, U.: Exper-

imental large-scale review of attractors for detection of potentially unwanted applications. Computers & Security **76**, 92–100 (2018)
25. Wireshark, `https://wireshark.org`
26. Xu, L., Veeramachaneni, K.: Synthesizing tabular data using generative adversarial networks. arXiv preprint arXiv:1811.11264 (2018)
27. Yuxin, D., Chen Sheng, Xu Jun: Application of deep belief networks for opcode based malware detection. In: Proceedings of 2016 International Joint Conference on Neural Networks (IJCNN). pp. 3901–3908. IEEE (2016)

8 Appendix I - List of Download URLs

Below is the list of websites used for downloading files for the experiment:

1. https://windows10portal.com
2. https://vlc-media-player.en.uptodown.com
3. https://www.mycleanpc.com/mwo/carts/
4. https://baidu-antivirus.en.lo4d.com/
5. http://pcclean.site/
6. http://www.driverupdate.net/download.php
7. https://www.bigfishgames.com/games/5489/midnight-castle/?pc
8. https://www.avira.com/en/free-antivirus-windows#start-download-fss
9. https://www.internetdownloadmanager.com/download.html
10. https://www.freedownloadmanager.org/
11. https://en.softonic.com/
12. https://download.cnet.com
13. https://notepad-plus-plus.org/download/v7.7.1.html
14. https://www.videolan.org/vlc/index.he.html
15. https://git-scm.com/downloads
16. http://www.angusj.com/resourcehacker/
17. https://www.mozilla.org/en-US/firefox/new/
18. https://www.chiark.greenend.org.uk/s̃gtatham/putty/latest.html
19. https://www.google.com/chrome
20. https://www.winzip.com
21. https://www.dropbox.com
22. https://www.telerik.com/download/fiddler
23. https://filezilla-project.org/download.php
24. https://keepass.info/download.html
25. https://openvpn.net/community-downloads/
26. https://pidgin.im/download/windows/
27. https://desktop.jitsi.org/Main/Download
28. https://tortoisesvn.net/
29. https://www.malwarebytes.com/
30. https://www.whatsapp.com/download/
31. https://www.skype.com
32. https://www.filehippo.com
33. https://www.sandboxie.com
34. https://www.jetbrains.com
35. https://www.scootersoftware.com
36. https://filezilla-project.org
37. https://openvpn.net
38. https://wireshark.org
39. https://7-zip.org
40. https://hex-rays.com