PUA Detection Based on Bundle Installer Characteristics

Amir Lukach, Ehud Gudes, Asaf Shabtai

PUA Detection Based on Bundle Installer Characteristics

Amir Lukach[1], Ehud Gudes[1,2], and Asaf Shabtai[3]

[1] Department of Computer Science, The Open University, Israel
[2] Department of Computer Science, Ben-Gurion University of the Negev, Israel
[3] Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Israel
amirluckach@gmail.com

Abstract. Many applications, such as download managers, antivirus, backup utilities, and Web browsers, 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 propose a machine learning-based method for detecting PUAs. Our approach can be applied on the target endpoint directly and thus can provide protection against PUAs in real-time.

Keywords: Potentially unwanted applications, Machine learning, Antivirus.

1 Introduction

Today, 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 [3].

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.* [3] 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,' and '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 collected 771 applications (96 of them are PUAs) from 40 different websites, and monitored their behavior during installation on a Windows 7.0 OS endpoint, on which an antivirus was 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, it was labeled as 'PUA,' otherwise it was labeled as 'benign.' We also implemented a C++ Windows application, which can monitors 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 all PUA detection based on bundle installer characteristics of the PUAs in our experiment, while maintaining a rate of only 0.5% false positives. The same classifier could detect 79.35% of the PUAs with zero false positives. Based on these results, we conclude 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 [6], 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. We implement and evaluate our proposed method on a dataset of 96 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.

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 the figure).

Profit is the motivating factor for the affiliates and PPI networks that spread both benign and unwanted software. A typical PUA installation scenario will

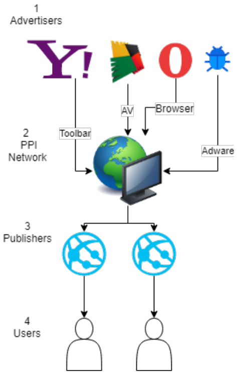

Fig. 1: The key players in the PPI business model.

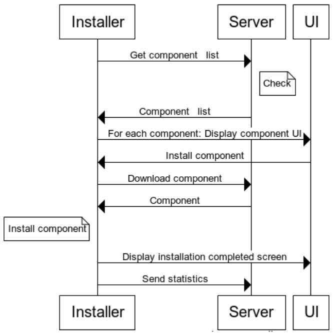

Fig. 2: PUA installation sequence diagram.

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 (2,1)). 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 pay per install model motivates application distributors to bundle several third party applications with their own application in return for a fee, which usually ranges from 0.01 to 1.5 USD, for each application installation. The payment to the PPI network by the advertisers is sometimes made up front, in return for a fixed number of installations. 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,[4] in the second quarter of 2019 there were 19 million PUA installations. Note that only 24% of the scans mentioned in the Quick Heal Threat Report were performed in real-time (there are other types of scans, e.g., an on-demand scan), and the report does not provide an estimate of the number of PUAs that go undetected. In order to avoid detection by Safe Browsing and other security programs, affiliates churn through domains every few hours or

[4] quickheal2019

actively cloak against Safe Browsing scans. PUAs require user consent to install additional applications, in order to avoid being marked as malicious by security programs. The PPI model described above is used in Section 4 when we discuss our PUA detection methodology.

3 Related Work

In the most relevant paper on PUA detection [7], 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 [9]. 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.* [6] 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.* [1] 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.* [3] 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 send a windows event to the currently displayed control. Stavova *et al.* [11], 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 [11]. In our study, we use some of the features proposed in [7] (see Section 4). Another study discussed the detection of PUAs in a mobile environment [12], mentioning key PUA indicators, such as functionality to root or jailbreak a device, remote monitoring, cracked or repackaged apps, and excessive advertisement packages. Since in our research we focus on desktop computers and the Windows environment, most of the features identified by [12] are not relevant for us.

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.* [2] 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 [8]. 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 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 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, and use a sandbox which can be evaded by a PUA, due to their similarity to benign applications. In addition, many automated malware analysis sandboxes can be detected by taking advantage of the artifacts that affect their virtualization engine [12]. In contrast to techniques commonly used for malware detection, our methodology uses different run-time characteristics (some of which were mentioned in [7]), e.g., whether the process is signed, continuously creates new window pop-ups, or uses Internet Explorer Control to display dynamic Web content.

4 The Methodology

We present a unique, yet practical method that allows users or antivirus 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 run-time, 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 performed 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.

A machine learning classifier based on bundle installer characteristics. 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 [7]):

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. Uses IE control (uses Internet Explorer Control in an MFC dialog) – a Boolean feature, 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.
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.

Note, that a total of six 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) read EXE signature, (2) check if EXE requests static or run-time admin permissions, (3) read EXE Import table, (4) register shell events (for file system changes notifications) and WMI process creation events to Windows, (5) check if EXE listens to a port, (6) check if EXE uses IE browser control, and (7) call 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 Experimental Evaluation

5.1 Dataset

Our dataset contains 771 files (675 benign samples and 96 PUA samples), downloaded from 40 different websites , including the most popular download sites [3]: filehippo.com, cnet.com, and softonic.com. In our experiment, we also tested bundle installers (e.g., Adobe Flash Player, which installs two security products, Avira AV, which is bundled with Opera Web browsers, and other applications we downloaded from commonly used download portals).

Each file was labeled as benign or PUA according to the following process: 1. Scanning in VirusTotal. Our ground truth is that a sample that received one or more positive classifications from significant engines of VirusTotal is a PUA. 2. Manually testing each application. This was done since we saw that in some cases, when downloading a PUA from a dedicated site a second time (two weeks later), we often obtained a clean scan for the newly downloaded PUA in VirusTotal for a PUA which was previously detected.

In addition, each application was downloaded to a Windows 7.0 endpoint, in which an antivirus is installed, using a Chrome browser which checks if the EXE is labeled as malicious/PUA using the Safe Browsing API. We ran each sample on the host computer itself and monitored it using process explorer and our monitor application. Two weeks after scanning each EXE, we scanned them again by downloading them from the same portals and other download sites, to understand the effectiveness of AV for PUA detection.

5.2 Experimental Setup

The applications were downloaded from January to October 2019; during this time, 771 Windows applications were downloaded using Chrome and installed on Windows 7 OS, in which AV was installed. The installed applications included the categories shown in Figure 3 which shows the frequency of each category.

AVs and browsers were downloaded from the vendor sites (e.g., Chrome was downloaded from google.com/chrome); the same applies for different utilities,

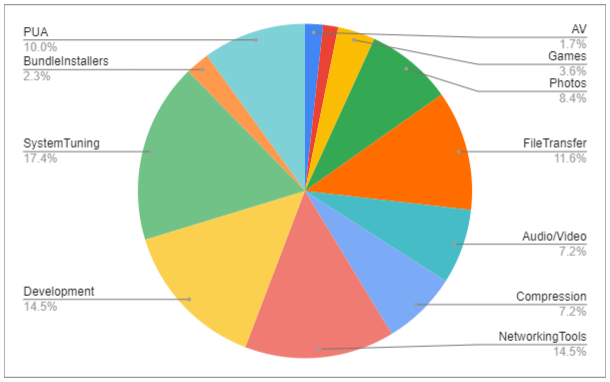

Fig. 3: Categories of installed applications.

e.g., PDF reader was downloaded from adobe.com. PUAs were downloaded by searching for free products, and they were also downloaded from a list of top ranked freeware websites: cnet.com, filehippo.com, and softonic.com [3].

5.3 VirusTotal's PUA Detection

VirusTotal uses 68 different AV engines. Note that when we began the experiment we used VirusTotal scans to determine whether the executable is a PUA or benign file. Two weeks later, we downloaded the PUAs again, and surprisingly 5% of the marked PUA, were not labeled as a PUA anymore (because the EXE file was modified by the hosting site). Some of the websites we used to download PUAs were blocked or the EXE was not found in a related portal. Furthermore, almost all of the PUAs were undetected by most commonly used AV engines. On average, there were 4.7 labels per PUA, with a median of five. Since only one AV is usually installed on a computer, this shows that in many cases, a PUA will not be detected by an AV. This also happened in our experiment, in which the installed AV did not detect any of the PUAs.

5.4 Classifier Detection Rates for PUAs

We evaluated three types of classification algorithms: rotation forest, decision tree, and naive bayes. The results are labeled in Table 1. Our Monitor application, uses the rotation forest classifier to detect if an application is PUA or benign. We used 10-fold cross validations for splitting the samples, to training and test data. The Rotation Forest, which was found to be the best classifier, produced 0.5% false positives for a true positive rate of 100% (i.e., all the PUAs were detected). When set to produce 0.0% false positives, the Rotation Forest classifier detected 79.5% of the PUAs. Detection errors can occur when applications bundle many applications, and one installer, installs multiple applications.

5.5 Research Limitations

Our work focuses on PUA executables, which are mainly downloaded from popular download sites [3]. There is a risk that these websites may contain PUAs from a limited number of publishers that they trust. In such a scenario, there might be bias in the classification of the PUAs' features, however these are still the most popular download sites, so the insights from this research are still useful for preventing PUA distribution. Moreover, most PUAs were downloaded from an IP address in Israel (although some were downloaded from an IP address in the US), and Israel is not a country, in which users are used to paying for software, for windows, like the US or UK [6]. Therefore, in US, for example, the classification between PUAs and benign applications would be even more distinct, since more applications would be installed, resulting in more created processes and folders.

6 Conclusions and Future Perspectives

6.1 Adversarial Machine 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 [4]. Behavior-based detection models are currently being investigated as a new methodology for defeating malware. This kind of approach typically relies on system call sequences to model a malicious specification/pattern. There is a new class of attacks, namely shadow attacks, which are used to evade existing behavior-based malware detectors by partitioning one piece of malware into multiple shadow processes. None of the shadow processes contain a recognizable malicious behavior specification known to single process-based malware detectors, yet as a group, these shadow processes can fulfill the original malicious functionality. Attackers are always searching for new attack vectors, and PUAs may serve as means of implementing shadow attacks. Furthermore, we can look at possible ways that adversaries can bypass detection by our method and their possible impact on affiliate network incentives and revenues, and their use of PUA in general. Such ways include:

1. Not running as administrator. This is very challenging, since the 'program files' folder requires administrator permission in order to modify/create files.

Table 1: Results

Classifier	FPR (TPR=1.0)	TPR (FPR=0.0)	AUC
Rotation Forest	0.005	0.795	0.99
Decision Tree	0.005	0.65	0.96
Naive Bayes	0.02	0.2	0.98

2. Reducing the number of processes. This is easy, but it means installing less components, since PUAs only bundle existing components, which by default are installed to different folders; this means that each installation will yield less money, which might affect the profitability of the PPI business model. Moreover, it is possible to create delayed system tasks to install more applications at a later time, however this reduces the chance of success, and it will be difficult to monitor whether the application was installed or not.
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 installation.

Furthermore, there is another method, that adversaries can use to bypass detection, which is code obfuscation techniques for evading malware detection, suggested by Rozenberg *et al.* [5], 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. These will not create a problem, for the proposed solution, since our monitor application monitors file system changes, created processes, and process privileges, and does not monitor or inspect specific system calls or API usage. 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.

6.2 Summary

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 [7]. Previously suggested methods, includes using sandbox [3]. Alternatively, trying to use sites like Virus Total, 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, if the PUA has not been scanned by their sites. Therefore, it is not a real-time solution. 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,' and 'number of folders created.' Such features distinguish PUAs from benign applications. 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. Solving this, is out of the scope of this paper, however, since we know when PUA was running, user can return to a safe state, by reverting to a windows restore point, before the PUA was installed. The proposed method can be used in order to reduce the distribution of PUAs and prevent their installation, since even if statistically a PUA is able to avoid detection by our method, our method's detection capabilities can reduce the distribution's profitability and weaken the PUA business model [6]. Our plans for future work include extending the experiment to a larger set of applications and applying similar methods on other operating systems (e.g., macOS).

References

1. 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)
2. Ding Yuxin, 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)
3. Geniola Alberto, 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)
4. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: European Symposium on Research in Computer Security. pp. 62–79. Springer (2017)
5. Ishai Rosenberg, Ehud Gudes.: Bypassing system calls-based intrusion detection systems, concurrency and computation. Practice and Experience 29 (2017)
6. Kotzias, P., Caballero, J.: An analysis of pay-per-install economics using entity graphs. WEIS (2017)
7. Mo, J.: How to identify pua (2016), https://www.infosecurityeurope.com/__novadocuments/86438?v=635670694925570000
8. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K.: Malware detection by eating a whole exe. In: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence (2018)
9. Rivera, R., Kotzias, P., Sudhodanan, A., Caballero, J.: Costly freeware: a systematic analysis of abuse in download portals. IET Information Security **13**(1), 27–35 (2019)
10. Rodrguez, J., Kuncheva, L., Alonso, C.: Rotation forest: A new classifier ensemble method. IEEE transactions on pattern analysis and machine intelligence **28**, 1619–30 (2006)
11. Stavova, V., Dedkova, L., Matyas, V., Just, M., Smahel, D., Ukrop, M.: Experimental large-scale review of attractors for detection of potentially unwanted applications. Computers & Security **76**, 92–100 (2018)
12. Svajcer, V., McDonald, S.: Classifying puas in the mobile environment. In: Virus Bulletin Conference (2013)
13. Thomas, K., Crespo, J.A.E., Rasti, R., Picod, J.M., Phillips, C., Decoste, M.A., Sharp, C., Tirelo, F., Tofigh, A., Courteau, M.A., 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)