Technical Disclosure Commons

Defensive Publications Series

February 13, 2019

Leveraging app relationships and distribution patterns to identify malicious software

Florian Tramer

Mo Yu

Sai Deep Tetali

Leveraging app relationships and distribution patterns to identify malicious software

ABSTRACT

Software distributors, such as the operators of online software repositories or stores, scan and analyze the software they host to flag potentially harmful applications (PHAs). The scans are typically performed offline and are based solely on app-level features and do not take into account structural relationships between different apps and devices. This disclosure describes an app ecosystem-based approach to detect PHAs via analysis of contextual information, such as app install statistics and installation distribution patterns. Relevant contextual information about each app obtained user permission is leveraged to build a machine learning pipeline to flag PHAs for further review. The ecosystem-based approach makes it difficult for malicious actors to evade detection. The techniques can be applied online at app install time and are complementary to detection mechanisms that involve direct analysis of apps.

KEYWORDS

- Malware detection
- Mobile app
- Harmful app
- Malicious app
- Install pattern
- App store
- Software download
- App distribution

BACKGROUND

Malicious parties that are successful in getting their software installed on user devices attempt to perform operations harmful to users, such as fraud, tracking, stealing sensitive data, etc. Such harmful software is generally referred to as malware. Software distributors, such as the operators of online software repositories or stores, scan and analyze the software they host to flag potentially harmful applications (PHAs).

The decision to flag an app as PHA is based on a collection of scores, obtained via specific forms of analyses, e.g., static and dynamic analyses, code similarity to known PHAs, signatures, etc. The scores can be combined by training large logistic regressions or deep neural networks. Such techniques are based solely on app-level features and do not take into account structural relationships between different apps and devices, such as co-installations, device models, etc. Moreover, the app-level techniques are applied offline and do not involve install-time detection of PHAs.

DESCRIPTION

This disclosure describes techniques to detect PHAs via in-context analysis of software on a user device, such as a smartphone, tablet, etc. The techniques are implemented with user permission. When the user permits, contextual information pertaining to relationships between apps or user devices is used to enhance malware detection. If the user permits, contextual information and metadata about apps on the user device is obtained, e.g., via periodic scanning, at the time an app on the device is installed or updated, etc. The information can include statistics for each app, such as number of installs, fraction of installs as a system application, the app that installed or originated the app, time of install, etc., as well as distributions across the device, such as the number of co-installed apps, number of co-installed PHAs, fraction of devices where the app was installed or originated by a PHA, etc. These app statistics and distribution patterns are analyzed to identify PHAs.

For a user device, the list of installed apps along with the relevant contextual information about each app installation event is leveraged to build a machine learning pipeline on top of app co-installation statistics to identify apps that are most likely to be malware. The detection of PHAs is based on the observation that co-installation of apps on a given device can be interpreted as a measure of similarity or complementarity between apps.

For each app, a histogram of all co-installs with different apps, such as malware, off-market (e.g., not obtained from a device manufacturer or OS provided official download platform), market-purchased (e.g., obtained from a device manufacturer or OS provided official download platform), system (e.g., pre-installed prior to sale of the device), etc., is computed. The histogram is converted to the cumulative distribution function (CDF) in order to facilitate processing by a machine learning model. Similarly, all distributional features in the contextual data obtained from the user device are converted to respective binned CDFs. The binned CDFs are used to train a machine learning model based on random forests to find apps that have similar distribution patterns as known malware. The apps that exceed the model thresholds for high confidence predictions are flagged as PHAs and are passed for further review, e.g., manual review by security experts, to confirm whether the classification is accurate.

For example, one of the inputs to the machine learning model can be based on the average fraction of PHAs among co-installed apps - the average of the ratios of PHAs among all other apps on a device for each device on which an app is installed. The ratio is compared to a threshold value to determine whether an app is likely a PHA.

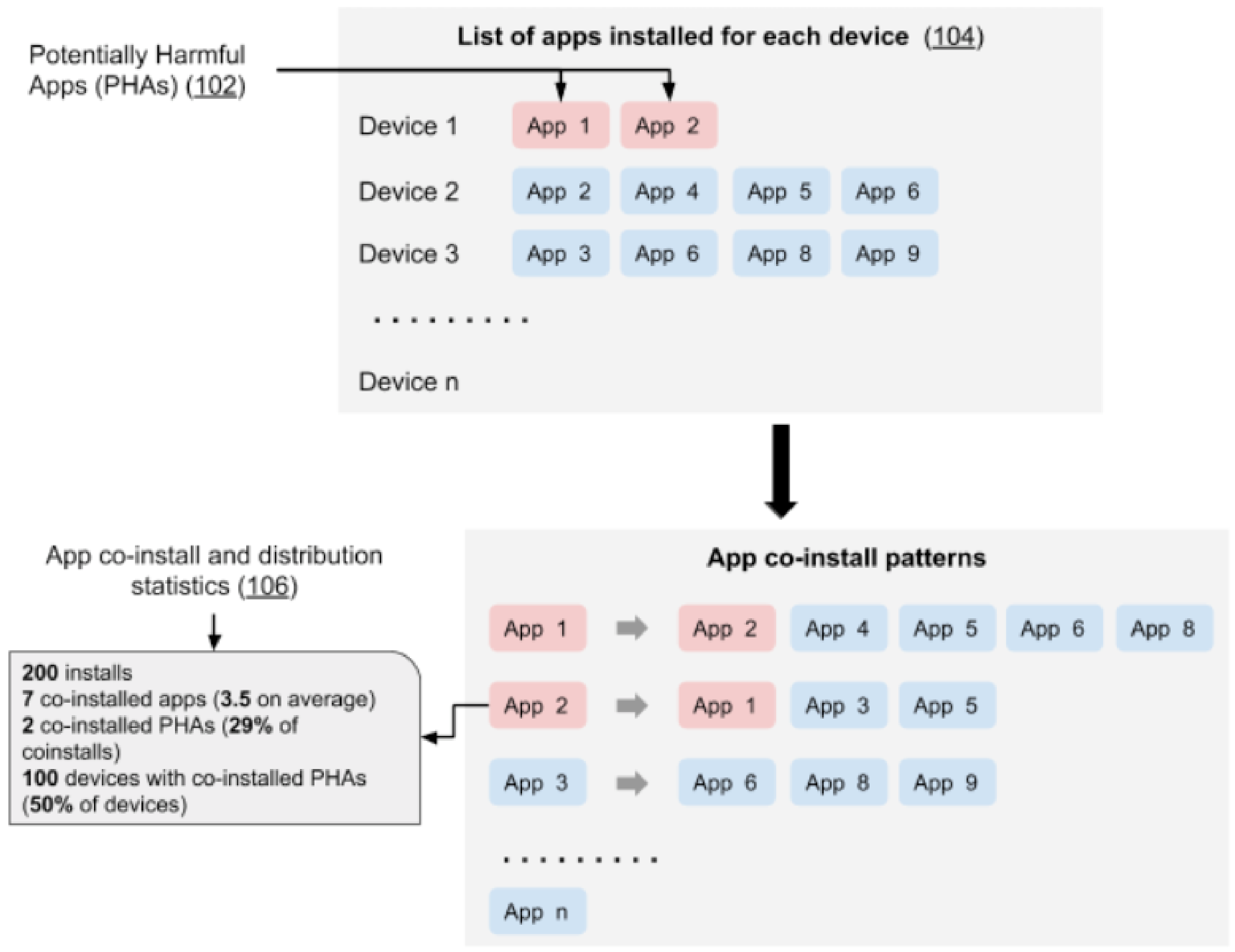
Potentially Harmful Apps (PHAs) (102)
List of apps installed for each device (104)
Device 1
App 1
App 2
Device 2
App 2
App 4
App 5
App 6
Device 3
App 3
App 6
App 8
App 9
Device n
App co-install and distribution statistics (106)
200 installs
7 co-installed apps (3.5 on average)
2 co-installed PHAs (29% of coinstalls)
100 devices with co-installed PHAs (50% of devices)
App co-install patterns
App 1
App 2
App 4
App 5
App 6
App 8
App 2
App 1
App 3
App 5
App 3
App 6
App 8
App 9
App n

Fig. 1: PHA detection based on app co-installation and distribution information

As an operational example of the application of the described techniques, Figure 1 shows a list of apps installed on various user devices, e.g., devices 1-n, each with respective apps installed. One of the three devices (Device 1) contains two PHAs (102), illustrated in red. With user permission, the list of apps installed on each device (104) is obtained. Lists obtained from multiple devices are combined to generate co-install and distribution statistics (106) about each app in the combined list of installed apps.

For instance, Figure 1 shows that app 3 is installed on 200 devices with 7 co-installed apps, 2 of which are PHAs and that 50% of devices on which app 3 is installed also has a PHA co-installed. The app co-install and distribution information for apps installed on all devices serves as one of the inputs to the trained machine learning model to compare against one or more relevant threshold values. Based on the comparisons, it is determined whether an app should be flagged as PHA.

Further, the source of installation of an app can also be used to detect PHAs. For example, consider a situation where it is known that an app triggers the installation of two other apps, one PHA, and one non-PHA. The other (non-PHA) installed app is also installed on other devices where the installation is triggered by a non-PHA app which itself was installed by another non-PHA app.

For each app installed on the two devices, the installation information is used to compile relevant statistics regarding the source of the installation. For instance, it can be determined that a particular app is present on multiple devices and was installed by a PHA on at least a subset of the devices. Further, it can be detected that the app did not trigger the installation of another app on any device. Similar to the earlier operational example illustrated in Figure 1, the app installation source information for apps installed on all devices serves as an input to the trained machine learning model to compare against one or more relevant threshold values. Based on the comparisons, an app is flagged as PHA or not.

The techniques of this disclosure are complementary to detection mechanisms that involve directly analyzing apps. In contrast to PHA detection schemes that depend on direct analysis of apps, the described techniques provide a detection mechanism that applies across the entire app distribution and installation ecosystem, using data obtained from users that provide permission. PHA detection via the described approaches is difficult to evade because it is based on an overall picture of the ecosystem which is resistant to external influence as it is difficult for any single malicious party to affect global statistics across all devices.

The application of machine learning within the approach adds further resistance to evasion and adaptation since neither malware authors nor their pursuers have knowledge of how the PHA detection system works. While apps can attempt to evade detection by the described techniques by obfuscation and encryption, these evasion approaches are likely to fail because they do not affect app distribution patterns that are used by the described techniques for PHA detection.

Moreover, unlike approaches that involve direct app analysis, the described techniques can enable online application of PHA detection at the time of app installation by using data obtained from user devices. Such online operation based on examining contextual app information obtained with user permission is particularly useful because user devices often lack the resources to use other malware detection techniques, such as static or dynamic app analysis. Further, network constraints often prohibit the possibility to obtain the whole app for off-device analysis.

The techniques of this disclosure can improve the recall of PHA detection while maintaining high precision, and therefore provide better protection against malware threats. The described machine learning pipeline for PHA detection can also enable lead to earlier detection of PHAs, e.g., compared to analysis-based approaches.

The techniques described above can be extended by examining additional contextual inputs, when user provide permission to access and use such data. The techniques can also be enhanced by the application of additional machine learning techniques besides random forests. The threshold values used by the machine learning model can be specified by relevant parties, such as app distribution platform owners, device manufacturers, users, etc. or can be determined dynamically.

Further to the descriptions above, a user may be provided with controls allowing the user to make an election as to both if and when systems, programs or features described herein may enable collection of user information (e.g., information about a user’s social network, social actions or activities, profession, a user’s preferences, or a user’s current location), and if the user is sent content or communications from a server. In addition, certain data may be treated in one or more ways before it is stored or used, so that personally identifiable information is removed. For example, a user’s identity may be treated so that no personally identifiable information can be determined for the user, or a user’s geographic location may be generalized where location information is obtained (such as to a city, ZIP code, or state level), so that a particular location of a user cannot be determined. Thus, the user may have control over what information is collected about the user, how that information is used, and what information is provided to the user.

CONCLUSION

This disclosure describes an app ecosystem-based approach to detect PHAs via analysis of contextual information, such as app install statistics and installation distribution patterns. Relevant contextual information about each app obtained user permission is leveraged to build a machine learning pipeline to flag PHAs for further review. The ecosystem-based approach makes it difficult for malicious actors to evade detection. The techniques can be applied online at app install time and are complementary to detection mechanisms that involve direct analysis of apps.