International Journal of Engineering and Management Research
www.ijemr.net

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-12, Issue-6, (December 2022)
https://doi.org/10.31033/ijemr.12.6.9

Malware Sandbox Evasion Techniques in Mobile Devices

Yugandharee Sankaranarayanan[1], Sarangan Ravindran[2], Suhail Ahamed[3] and Kajanthan Balendraraja[4]
[1]Faculty of Computing, Cyber Security, SLIIT, Colombo, SRI LANKA
[2]Faculty of Computing, Cyber Security, SLIIT, Colombo, SRI LANKA
[3]Faculty of Computing, Cyber Security, SLIIT, Colombo, SRI LANKA
[4]Faculty of Computing, Cyber Security, SLIIT, Colombo, SRI LANKA

[1]Corresponding Author: it19017884@my.sliit.lk

ABSTRACT

The mobile platform is where it's at. There are currently very few professionals who dispute this view. Because of the rapidly increasing number of smartphones and other devices powered by the Android operating system all over the world, there has been a corresponding surge in the number of mobile apps, particularly harmful mobile apps. This form of malware is very new, but it is rapidly changing, and it brings hazards that have not been seen before. As a part of Check Point's ongoing efforts against the rising tide of mobile dangers, we, the Malware Research Team, want to learn as much as we can about the constantly shifting Android malware landscape. This requires understanding the internal operation of as many malicious apps as we can, so we can learn as much as we can. Manual malware analysis has always been a difficult operation, taking days or even weeks to complete for each sample. Because of this, the work is impracticable even for a small sample pool because of the amount of time it takes. Following the successful application of this strategy to mobile malware, our response is to automate as much of the analysis process as is practically practicable. Idan Revivo and Ofer Caspi from Check Point's Malware Research Team were tasked with developing a system that would take an application and produce a report describing exactly what it does when it is run, specifically pointing out anything "fishy." This would enable us to perform an initial analysis with no human intervention, which is exactly what they have done. The popular CuckooDroid sandbox and a few other open-source projects form the basis of this automated, cross-platform emulation and analysis framework, which allows for static and dynamic APK inspection in addition to evading some VM-detection techniques, encryption key extraction, SSL inspection, API call trace, basic behavioral signatures, and more. It is easy to make changes and add new features to the framework, and it draws heavily on the expertise of the current Cuckoo community.

***Keywords*--** Malware, Android, Sandbox, Security, Mobile

I. INTRODUCTION

Hackers are employing the latest technology to overcome defenses, making cyber assaults more challenging by the day. After all, the only thing that counts to a virus creator is that the product remains undetectable. Whenever malware enters into touch with various protective and analysis engines, such as a sandbox and an anti-virus, it must be kept hidden and unobtrusive at first. A sandbox is a network-based segregated workspace that simulates end-user operating models. Sandboxes are used to execute suspect programs without putting the host device or network at risk. Using a sandbox for sophisticated malware detection adds another layer of defense against emerging security threats, such as zero-day malware and subtle attacks. And what happens in the sandbox stays in the sandbox, preventing malfunctions and the propagation of software flaws. Sandboxes are divided into several types, including applets, jails, and virtual machines that run a guest operating system with restricted or rule-based access to system applications. Application sandboxes are the most common of these, as they allow dangerous applications to operate in a separate operating system without harming the host operating system. For the running and testing of malware binaries, there are various online and standalone sandboxes available. Anubis, Cuckoo, Malwr, ThreatExpert, Comodo Instant Malware Analysis, Joe Sandbox, FireEye Malware Analysis (AX Series), and TrendMicro Dynamic Threat Analysis System are just a few of the famous ones. In the future, sandbox-evading malware is expected to become a common powerful tool in the hands of hacktivists while ransomware and zero-day exploits were considered as big threats in the past decade. When considering mobile devices, there are fewer detecting practices to find the malwares like in computers. Lack of performance and precision on Built-in malware identification systems in android devices, and poor identification ability on exploit APKs and URLs which violates data Privacy mechanisms harvesting user data more than the required threshold. In this research, we are focusing on finding ways to use malware sandbox evasion techniques to detect the malware for android mobile devices.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-12, Issue-6, (December 2022)
https://doi.org/10.31033/ijemr.12.6.9

II. BACKGROUND AND LITRATURE SURVEY

A. Sandbox

Sandboxes are designed to test for malwares. Application can run in sandboxes, which are closed environments, under close supervision. They enable defenders to check both known malware and unknown software for dangerous behavior in order to develop behavioral signatures for use in anti-malware systems. In order to confine the undesirable consequences of potentially dangerous programs, these systems might be physical devices (sometimes referred to as "bare-metal sandboxes") with restricted network capabilities. Virtual machines and other system emulators, however, offer a scalable platform for building malware sandboxes because of the enormous quantity of program executables that have been analyzed.

As it was already said that the Android platform has a greater market area, the "Google Play Store," the major application marketplace on the Android, has almost 3 million applications accessible for download, and that figure is growing every day [1]. It is hard to manually analyze each app in light of the enormous surge of new ones. Malware sandboxes are used to automate the detection and removal of dangerous apps from the environment by application marketplaces and security companies.

B. CuckooDroid

CuckooDroid is a feature of Cuckoo Sandbox, an open-source program for automatically analyzing dynamic malware. It makes it possible for Cuckoo to run and examine Android applications.

Cuckoodroid has two major components: a "host" and a "guest." architecture of the CuckooDroid is shown in Figure 1.

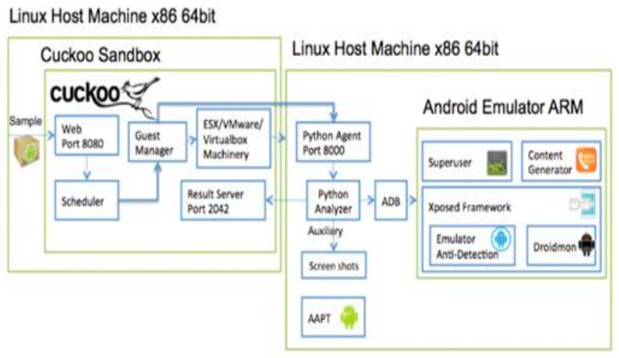


Figure 1: CuckooDroid Architecture

The android emulator is managed by Cuckoo Sandbox, which also creates a report at the end of the report. The application is executed by Android Emulator, which then gathers data from it and reports it to Cuckoo Sandbox. The description of a few of the major components in figure 1 is provided above.

C. Malware

One-third of mobile devices are at medium to high risk of data exposure, and Android smartphones are roughly twice as likely as iOS devices to contain malware. In this part, we will discuss some of the most common mobile malwares.

Trojans

A Trojan is a piece of software that seems to the user to be a harmless program but executes dangerous operations in the background. Trojans are employed to aid in the assault on a system by executing actions that may weaken the system's security, allowing for easy hacking. FakeNetflix is an example of a Trojan that harvests user credentials for Netflix accounts in Android settings. The Trojan KeyRaider was used to steal Apple IDs and passwords.

Root exploits - back doors

Backdoors employ root access to hide malware from antivirus software. Rage against the cage (RATC) is a common Android root hack that allows complete device control. If the root exploit achieves root power, the malware can conduct any activity on the device, including the installation of programs while the user is ignorant. Xagent is an iOS Trojan that opens a back door and grabs data from the attacked device.

Ransomware

Ransomware restricts users' access to their data by locking the device or encrypting the data files until the ransom is paid. Fake Defender is malware that masquerades as Avast antivirus. For the purpose of money, it locks the victim's device. In 2017, hackers exploited a Safari weakness used for pop-ups to create an iOS malware.

Botnets

A "bot" is a sort of malware that allows an attacker to take control of an infected mobile device. They are part of a network of infected computers known as a "botnet," which is generally made up of all victim mobile devices worldwide. Geinimi is a botnet for Android.

Spyware

Spying software is what spyware is. It runs in the background undetected while collecting data or providing remote access to its author. Android malware such as Nickspy and GPSSpy observes the user's sensitive information and sends it to the owner. Passrobber is an example of iOS Spyware since it is capable of intercepting outbound SSL traffic, checking for Apple IDs and passwords, and sending these stolen credentials to a command-and-control server.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-12, Issue-6, (December 2022)
https://doi.org/10.31033/ijemr.12.6.9

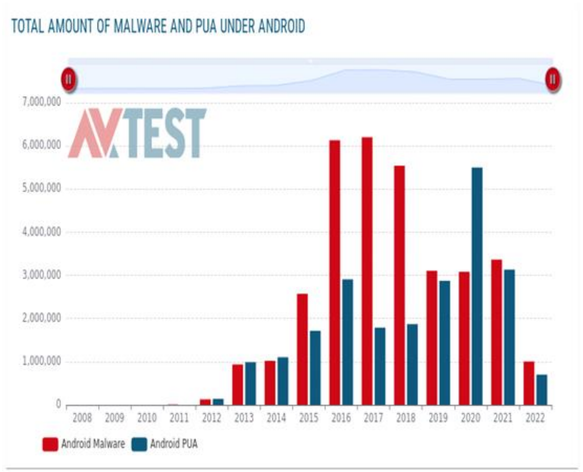


Figure 2: Android malware and PUA amount

D. Malware Evasion Techniques

In their 2016-year results Kaspersky LABs claimed that malware developers explored new methods to circumvent Android security systems. To evade discovery, malware authors must continually analyze mobile security solutions and develop new ways. These are known as evasion strategies, and they are detailed below.

Anti-security techniques:

These strategies are used to prevent detection by security devices and applications such as anti-malware, firewalls, and other environmental protection measures.

Anti-sandbox techniques

Sandboxing is a technique used to segregate operating applications and therefore avoid any harm to the computer system from untested apps. The anti-sandbox technology is used to identify automated analysis and to avoid reporting on malware activities. This is accomplished by recognizing registry keys, files, or processes associated with virtual environments.

Anti-analyst techniques

To avoid reverse engineering, these solutions employ a monitoring tool. To monitor and detect malware, analysts may use tools such as Process Explorer or Wireshark.

E. Malware Analysis Techniques

Malware analysis for android applications is often done in one of two ways: static analysis or dynamic analysis or combination of both. The static analysis examines several features without actually running the application. The manifest file needed by the application is a crucial asset that many frameworks examine. The Android Manifest contains meta data about the specific package name, utilized activities, services, broadcast receivers, and content sources. It identifies the classes that carry out these elements and makes their capabilities accessible. The Android operating system uses this knowledge to determine when each component must be launched. The manifest also specifies the permissions required to access the API's protected areas. Access to particular hardware elements may be a sign of malicious activities.

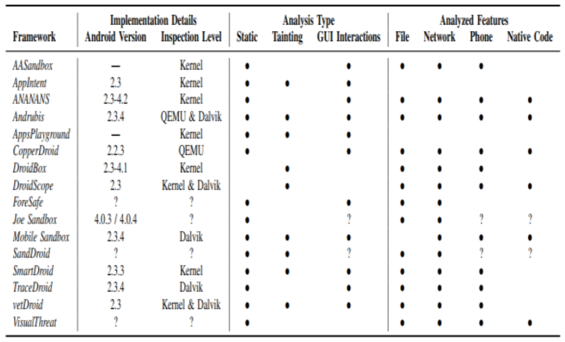
Analyzing the applications' byte code is another tactic. The application's possible pathways cannot be predicted because the code is not run and no variables are set. Analysts can comprehend an application's internal workings and the relationships between its code blocks with the aid of graphs [2]. With that method, suspicious API requests that access sensitive information can be found. API calls that encrypt or decrypt data or run external code are frequently used to obfuscate code, but they may also be found through static analysis [3]. Checking each type of resource file in the Android Application Package (APK) will reveal any external code. Malware frequently conceals libraries in external files that appear to be innocent in order to disguise suspicious API calls. Dalvik Executable Files (.dex Files) are created when Android apps are built. String searches are possible in the disassembled.dex files. IP addresses that potentially lead to command-and-control servers or data sinks for sensitive information can be found by scanning these strings for them. Androguard, which disassembles and decompiles Dalvik byte code to Java source code, is a well- known tool for static code analysis. That static code analyzer is used by frameworks including Sanddroid, Andrubis, and Tracedroid.

The program will run on either a virtual computer or an actual device as part of the dynamic analysis strategy. The analysis includes observing and analyzing the application's activity. Compared to the static analysis, the dynamic analysis produces a less abstract understanding of the application. Only few of the possible code pathways are actually used during runtime. High code coverage is the fundamental objective for analysis frameworks since all activities should be taken in order to detect any potentially dangerous activity. According to research, code coverage for fully randomized input is 40% or below [4]. Various methods exist to keep an eye on an application's behavior depending on the data of interest. Taint tracking is one analytical method. Message flow analysis and potential exploitation of private, sensitive information by third-party apps are both possible with a system-wide enabled taint propagation [6]. TaintDroid is a widely used framework that employs that method. It tracks the real-time access and manipulation of user data by apps and was created with the Dalvik Virtual Machine. While moving through variables, files, and messages, it marks the sensitive data. But TaintDroid can only identify explicit data flow; it cannot examine implicit flow through control flow. That channel could be used to send sensitive information [5].

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-12, Issue-6, (December 2022)
https://doi.org/10.31033/ijemr.12.6.9

F. Existing android malware analysis sandboxes

When it comes to sandboxes, there are a few frameworks, sandboxes, and analytic systems already available for android sandboxes, as well as some that have been proposed as well. Static approach was initially employed to assess Android apps. A rudimentary system that has been developed by Schmidt et al, utilizes the "readelf" program to extract the function calls from an Android application and compares the resultant list with the information of identified malware [6]. Another instance of the static analysis technique is "Androguard", a totally open-source system proposed by Desnos et al. In this scenario, the system decompiles the application and uses signature-based malware identification [7] [8]. It was discovered that malware authors began to develop more obfuscated code, which has demonstrated its efficacy against static analysis, indicating that static analysis alone is insufficient for those advancing malware. As a result, researchers developed a dynamic analysis mechanism for Android apps. The first solution with dynamic analysis that offers real-time analysis by utilizing Android's runtime environment is "TaintDroid" by Enck et al [5]. By Lantz [9], a completely automated user emulation and reporting system that goes by the name "Droidbox" was added to this system. "Droidbox" is a powerful tool for analyzing Android apps, however it doesn't have the ability to log native API calls. The very first system integrating static and dynamic analysis for the Android platform in a very primitive manner was the AASandbox system by Bläsing et al [10]. Sadly, it appears that AASandbox is no longer being managed. DroidRanger is a system developed by Zhou et al [11]. that combines static and dynamic analysis. DroidRanger uses a mix of permission-based behavioral foot prints to identify samples of existing well-known malware families and a heuristic-based filtering method to identify unidentified harmful groups.



Framework	Implementation Details		Analysis Type			Analyzed Features			
	Android Version	Inspection Level	Static	Tainting	GUI Interactions	File	Network	Phone	Native Code
AASandbox	–	Kernel	•		•	•	•	•	
AppIntent	2.3	Kernel	•	•	•				
ANANANS	2.3-4.2	Kernel	•		•	•	•	•	•
Andrubis	2.3.4	QEMU & Dalvik	•	•	•	•	•	•	•
AppsPlayground	–	Kernel	•	•	•				
CopperDroid	2.2.3	QEMU	•		•	•	•	•	•
DroidBox	2.3-4.1	Kernel		•		•	•	•	
DroidScope	2.3	Kernel & Dalvik		•		•	•	•	•
ForeSafe	?	?	•		•	•	•		
Joe Sandbox	4.0.3 / 4.0.4	?	•		?	•	•	?	?
Mobile Sandbox	2.3.4	Dalvik	•	•	•		•	•	•
SandDroid	?	?	•	•	?	•	•	?	?
SmartDroid	2.3.3	Kernel	•	•	•	•	•	•	
TraceDroid	2.3.4	Dalvik	•		•	•	•	•	
vetDroid	2.3	Kernel & Dalvik	•	•	•	•	•	•	
VisualThreat	?	?	•			•	•	•	•

III. METHODOLOGY

Our first journey into the world of study involved looking through various historical records and manuscripts. Because this strategy provided us with the ability to take into account the three key obstacles that were discussed earlier, we came to the conclusion that it was the best one for us to adopt. In order for us to carry out this automated research method, we made use of the resources that were provided by IEEE, Science Direct, Research Gate, and Medline (the version that is available through PubMed). The search was limited to articles in academic periodicals and journals that were exclusively available in the English language between the years of 2010and 2022. Additionally, the works had to have been evaluated by other specialists before they were published. Throughout the course of the investigation, a number of distinct search terms, such as "Analyzing the system information like CPU core count, Digital system signature, installed programs, OS reboots and hardware components,"CuckooDroid Sandbox status research papers," "Sandbox mobile application testing," "Android applications malware detection," and "Mobile applications malware testing," were utilized; these terms were all owned by us for the purpose of finding the research papers. In addition, the reference lists of studies that were included because they satisfied the criteria for inclusion were combed through in order to look for prospective research that might fit those criteria and be added. As a result of our investigation, we were made aware of a hole in the research, and we proposed that it be filled by improving the testing of the mobile applications troths CuckooDroid Sandbox that were developed in relation to the analyzed malware detection system that was developed by us. This realization came about as a direct result of our having discovered the hole in the research. The realization that there was a gap in our knowledge initially sparked the thought that we should do this. It is necessary to carry out these steps in order for the gap to be filled.

As the second step to examine it and test Android applications, the CuckooDroid sandbox first has to be installed. There are three options for installing this sandbox. Android on Linux Machine, Android Emulator, Android Device Cross-platform. Using the first approach, we have installed it. The data, which consists of android application files, was gathered from the online sources. We obtained around 30 samples from web surfing, and we also obtained about 3000 samples from the Canadian Institute for Cyber Security [12] by reading various study articles. Even though we obtained many samples, we were unable to test them all since CuckooDroid only supports Android 4.1.2, while our samples were unable to install on the emulator due to an outdated SDK version.

IV. RESULTS AND DISCUSSION

Obad.A and EvadeMe were the primary two applications we concentrated on. The older SDK version problem prohibits the EvadeMe Application from being deployed and tested in the CuckooDroid environment. However, a Virus Total scan was unable to identify the EvadeMe program as malicious.

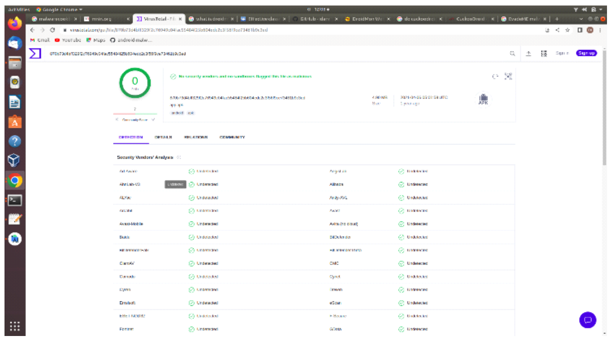


Figure 4: Virus Total Report of EvadeMe

Using the dex2jar tool, we converted the EvadeMe app into a jar file and used Java decompiler to examine it. The MainActivityKt class in that application has various methods for retrieving device information that may be used to verify the environment.

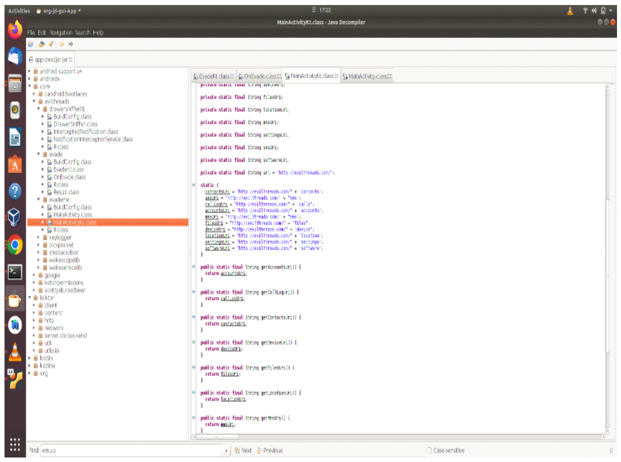
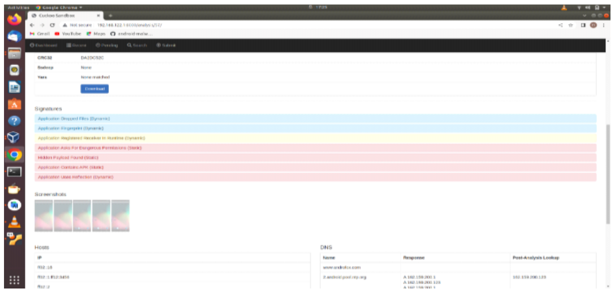


Figure 5: Snapshot of EvadeMe decomplication

CucukooDroid examined the "Obad.A" application and generated a report. The report includes information on permissions, signatures, fingerprints, and other facts, but the issue is that CuckooDroid's report does not mention of whether it is malware or not, unlike other analysis reports. Although it employs the colors green and red, it is difficult to determine if the app is harmful or not based just on those hues.



The testing results show that the CuckooDroid has excellent capabilities to deal with dynamically developing malware, but it relies on out-of-date "virus total" and "malware Cook book" information, making it unable to identify newly evolving malware. Additionally, identifying malware will become more important in the future, thus CuckooDroid's malware detection features, which may be used to find more advanced malware in Android applications, should be enhanced.

V. CONCLUSION & FUTURE WORK

Due to Android's status as the leading mobile operating system for smartphones, it has attracted the attention of researchers and malicious software developers alike. Despite the many proposed malware analysis methods, the number of malicious apps specifically developed to harm Android devices is growing at an alarming rate. Technologies like sandboxing are available for the detection such sophisticated malware but, modern malware will nearly always try to detect and circumvent a sandbox if one is present. When an application learns it is running in a sandbox, it may opt to avoid doing anything that could get it into trouble, such as deleting itself from disks, terminating, or using some other evasion technique.
In this research, we analyzed CuckooDroid, an Android malware detection tool that has several features, and the ways to improve it to recognize dynamically changing malware. Future work will test out simulated user behavior in sandbox environments, fake networks, change and share information about various system artifacts when malware requests it, to demonstration the CuckooDroid to the malware as a real environment and also improve the Cuckoodroid malware detection signatures that are already in place.

ACKNOWLEDGMENT

This research was supported by the authority of Sri Lanka Institute of Information Technology (SLIIT), and we would like to express our gratitude towards them.

e-ISSN: 2250-0758 | p-ISSN: 2394-6962
Volume-12, Issue-6, (December 2022)
https://doi.org/10.31033/ijemr.12.6.9

REFERENCES

[1] Kondracki, Brian, et al. (2022) The droid is in the details: Environment-aware evasion of android sandboxes. *Proc. Network and Distributed Systems Security Symposium (NDSS)*.
[2] Johannes Hoffmann. (2014). From mobile to security. *PhD Thesis, Ruhr-Universitt Bochum.*
[3] Arp, Daniel, et al. (2014). "Drebin: Effective and explainable detection of android malware in your pocket. *Ndss., 14*.
[4] Gilbert, Peter, et al. (2011). Automating privacy testing of smartphone applications. *Technical Report CS-2011-02.*
[5] Enck, William, et al. (2014). Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS), 32*(2), 1-29.
[6] Schmidt, A-D., et al. (2009). Static analysis of executables for collaborative malware detection on android. *IEEE International Conference on Communications*.
[7] Lilicoding, “Lilicoding/SA3Repo: A repository of peer-reviewed publications in the field of static analysis of Android apps,” GitHub. [Online]. Available at: https://github.com/lilicoding/SA3Repo. [Accessed: 18-May-2022].
[8] Desnos, Anthony & Geoffroy Gueguen. (2011). Android: From reversing to decompilation. *Proc. of Black Hat Abu Dhabi, 1*.
[9] “Droidbox – Android Application Sandbox,” The Honeynet Project. [Online]. Available at: https://www.honeynet.org/projects/active/droidbox/. [Accessed: 12-Jun-2022].
[10] Bläsing, Thomas, et al. (2010). An android application sandbox system for suspicious software detection. *5th International Conference on Malicious and Unwanted Software. IEEE*.
[11] Zhou, Yajin, et al. (2012). Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. *NDSS, 25*(4).
[12] “MalDroid. (2020). Datasets | Research | Canadian Institute for Cybersecurity | UNB,” www.unb.ca. [Online].Available:https://www.unb.ca/cic/datasets/maldroid-2020.html.