Computing (2021) 103:2643–2665
https://doi.org/10.1007/s00607-021-00954-6

REGULAR PAPER

Blockchain-based public ecosystem for auditing security of software applications

Qinwen Hu[1] · Muhammad Rizwan Asghar[1] · Sherali Zeadally[2]

Received: 15 October 2020 / Accepted: 27 April 2021 / Published online: 20 May 2021

Abstract

Over the years, software applications have captured a big market ranging from smart devices (smartphones, smart wearable devices) to enterprise resource management including Enterprise Resource Planning, office applications, and the entertainment industry (video games and graphics design applications). Protecting the copyright of software applications and protection from malicious software (malware) have been topics of utmost interest for academia and industry for many years. The standard solutions use the software license key or rely on the Operating System (OS) protection mechanisms, such as Google Play Protect. However, some end users have broken these protections to bypass payments for applications that are not free. They have done so by downloading the software from an unauthorised website or by jailbreaking the OS protection mechanisms. As a result, they cannot determine whether the software they download is malicious or not. Further, if the software is uploaded to a third party platform by malicious users, the software developer has no way of knowing about it. In such cases, the authenticity or integrity of the software cannot be guaranteed. There is also a problem of information transparency among software platforms. In this study, we propose an architecture that is based on blockchain technology for providing data transparency, release traceability, and auditability. Our goal is to provide an open framework to allow users, software vendors, and security practitioners to monitor misbehaviour and assess software vulnerabilities for preventing malicious software downloads. Specifically, the proposed solution makes it possible to identify software developers who have gone rogue and are potentially developing malicious software. Furthermore, we introduce an incentive policy for encouraging security

✉ Muhammad Rizwan Asghar
r.asghar@auckland.ac.nz

Qinwen Hu
qhu009@aucklanduni.ac.nz

Sherali Zeadally
szeadally@uky.edu

1 School of Computer Science, The University of Auckland, Auckland, New Zealand

2 College of Communication and Information, University of Kentucky, Lexington, USA

engineers, victims and software owners to participate in collaborative works. The outcomes will ensure the wide adoption of a software auditing ecosystem in software markets, specifically for some mobile device manufacturers that have been banned from using the open-source OS such as Android. Consequently, there is a demand for them to verify the application security without completely relying on the OS-specific security mechanisms.

Keywords Blockchain · Software audit · Security evaluation · Software security

Mathematics Subject Classification 68N01

1 Introduction

In the twenty-first century, our daily lives and work have gradually moved into the digital world. As a bridge connecting humans and machines, software plays an irreplaceable role in this revolution ranging from online shopping applications to digital office system and industrial control systems. We use different applications, and they play an important role in our daily lives. As we celebrate the efficiency which software brings in our lives, other concerns such as protecting security and privacy, and ensuring that the software would not snoop on user Identifications (IDs), passwords, or any private content, including photos, videos, and messages are also emerging. Looking back at the software development history over the past two decades, we find that attacks against software are never-ending. For instance, consider the LifeBridge health data breach incident [1] that occurred in September 2016 and compromised sensitive information of 539,000 patients along with employee-related information. The detection of Malicious Software (Malware) or securing software have been explored for many years, e.g., by using Hypertext Transfer Protocol Secure (HTTPS) to ensure information integrity between communicating parties, or antivirus software to prevent some attacks. However, such security mechanisms have not stopped software attacks altogether. For instance, a piece of code can be inserted malicious code by hackers during software distribution. In 2013, hackers [2] were able to compromise the patch management system and deployed forged patch files to the client. As a result, the integrity of the patch file was not verified before the client updated the patch. This security incident caused many clients to be infected and the computer network was paralysed. For example, in 2016, an Israeli security firm, Check Point [3] observed that more than 85 million devices (including Android and iOS) installed malicious apps. One such malicious app is 'Hummingbad'. The app steals banking information from the user's smart device. Android and Apple stores removed the app after security experts revealed the security issue. This security incident showed that the public software platform lacks software security validation. In 2019, Check Point discovered [4] that around 25 million Android devices had been affected by malicious apps 'Agent Smith'.

In Fig. 1, we demonstrate an example of how to upload malicious software to evade a software audit process, and how users get infected by downloading software from third party platforms. First, a Software Developer releases a software application

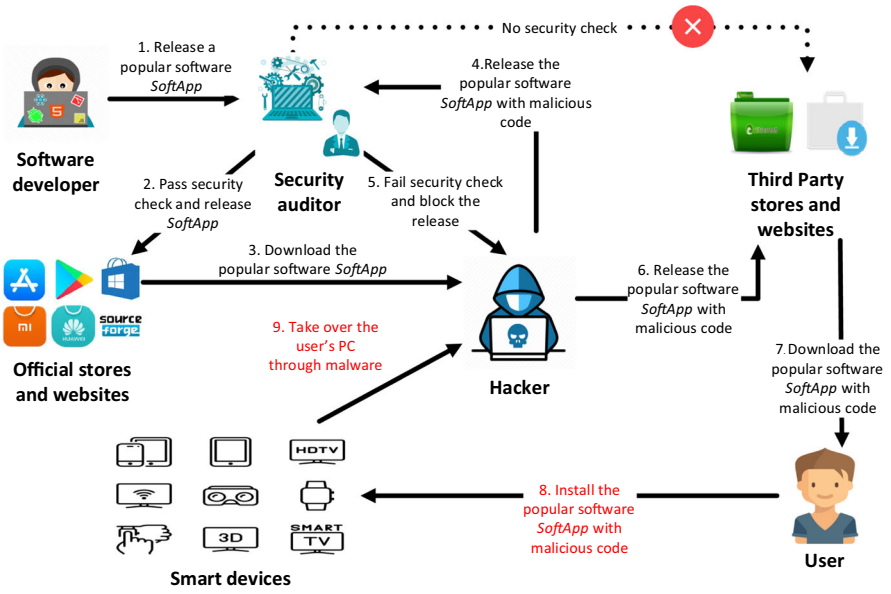

Fig. 1 Potential security risks of downloading software applications from third party websites or stores. The dotted line (having a cross sign) highlights that there is no regulatory mechanism between security auditors and third party websites or stores for assessing the software security level

SoftApp and generates an integrity value—which could be a checksum, say Message Digest 5 (MD5) or Secure Hash Algorithm 3 (SHA-3)—of this new version (Step 1). Next, after a Security Auditor determines that *SoftApp* has no security risk, the auditor uploads *SoftApp* to the Official Stores and Websites (Step 2). Next, a hacker downloads *SoftApp* from one of the official platforms (Step 3), and tries to upload *SoftApp* with malicious code (Step 4). However, the security auditor detects some security risks in the modified *SoftApp*. As a result, the security auditor prevents *SoftApp* from being uploaded to the official platforms (Step 5). Although the hacker cannot upload the modified *SoftApp* to official websites or stores, the hacker can still upload the modified *SoftApp* through third party platforms (Step 6). Users can choose to go to the official platforms to download the software but they can also choose to download it from third party platforms. However, for a variety of reasons, some users are willing to choose third party platforms to download any software application. For instance, geographic restrictions might be a major obstacle because some stores are not available in all countries (e.g., Google Play Store). So, many users in those countries would rely on third party platforms to download and install software applications. Second, users may want to install an older version of the software due to potential issues in the upgraded versions, or the new version is not compatible with older Operating System (OS). Third, some applications for Windows and Mac sometimes cost much money. However, it is likely in most cases that users might be able to get those applications for free (or at a lower price) from third party platforms. All in all, a user may choose a third party platform to download those software applications for one of the aforementioned reasons. In Fig. 1, the user downloads *SoftApp* with malicious code from third party

websites or stores (Step 7). Then, the user installs *SoftApp* on a device (Step 8). The result is that the hacker can take control of the user's device by using malicious code injected in *SoftApp* (Step 9). Figure 1 shows that users cannot ensure the security of software if downloaded from the third party platforms. Recent security incidents [4,5] suggest that online software platforms have loopholes when it comes to software security checks. Unfortunately, the auditing procedure between security vendors and online software stores has not been established. Thus, if hackers are able to compromise an online store, then they can easily launch the attack, which has been mentioned in [2]. The checksum mechanism will fail as the same application could generate different binaries and be uploaded to different software platforms. Hence, the user has no way to verify which hash is the checksum of the application, and which ones have been modified. Furthermore, there is no solution that can trace the history of software release and ensure the reliability of software release information. Although the devices can use the digital signature to ensure the safety and authenticity of the software that they deal with, a recent attack [6] demonstrated the feasibility of using faking digital signatures to bypass the system security checking. Besides, antiviruses are very difficult to detect zero-day attacks [7]. The Single Point of Failure (SPoF) issue has been ignored in the existing security audit process. To sum up, we need a novel solution to solve the following two issues: first, ensuring that the software has not been modified if a user downloads the software from a third party website; second, how to detect malware, if a user downloads it from a third party website or directly receives from a hacker?

Research contributions In this work, we present a novel auditing software ecosystem, which is based on blockchain technology for ensuring transparency, traceability, and integrity. By using blockchain technology for the software management platform, we can ensure when a software developer uploads software to online stores, all the participants in the network communicate with each other using a pre-determined mechanism to check that the software is valid. More specifically, we make the following research contributions:

- The proposed ecosystem enables users to compare software release information contained in the downloaded application using the information retained in the ecosystem. If the information is inconsistent, the user can check with security auditors to audit the downloaded application.
- The proposed ecosystem allows users to generate or get a report for malware. As a result, network participants can query the security level of the downloaded application. Moreover, security auditors participate in this network to help end users to assess software vulnerabilities.
- We propose a credit system for software developers, stores, and websites. The end users can judge the credibility of software-based developers' or websites' ratings. Furthermore, we use virtual credits to encourage users to participate in this ecosystem and maintain the quality of verification results.
- We provide a platform for users to actively inquire whether the software is reliable or not. Moreover, we design a public community for users to warn other users not to download malware. Most importantly, in order to prevent users from providing false information, we propose a voting mechanism for using security auditors to distinguish between malware and benign software applications. Using our pro-

posed platform, the existing software online stores can connect to the blockchain network to share and extract the software security assessment information from the block. Also, the proposed solution effectively monitors the software management process. It also reduces a single point of failure problem and improves security.

The rest of this article is organised as follows. Section 2 reviews related work. Section 3 describes our methodology and use case scenarios. In Sect. 4, we discuss potential security risks of using the proposed solution along with some performance aspects. Section 5 provides security analysis and explains how to configure the system to improve the performance. Section 6 concludes this article and provides research directions for future work.

2 Related work

Over the last two decades, malware detection has been receiving increasing interests. Research efforts [8–12] range from verifying computer-based malware to discovering malware in mobile devices. Many researchers proposed solutions [9,12–14] to analyse the application files as well as monitoring the behaviour of software at runtime. While other studies [15–17] utilise the intelligence of deep learning technologies to learn from known malware samples, and create a predictable learning model for effective detection of malware. Furthermore, the use of sandboxing and OS blocking technology [18–21] has also been widely used by software platforms in the past decade to discover malware software.

2.1 Static and dynamic analysis

There are two common approaches to detect malware: static and dynamic analysis. The static analysis refers to the solution that analyses application files to find possible malware without executing it. For instance, inspired by [22,23], Chen et al. [8] proposed a gene-inspired malware analysis approach. They used a static decompiling tool to disassemble the data of windows executable files. The proposed solution detects applications that have similar genes and categorises them into the same classes. This method allows users to build up the malware genes classes based on known malware samples. As a result, if the software has a similar gene to one of the malware classes, the system can then detect that the software may be vulnerable. Elish et al. [10] developed a static analysis method that classifies the dependence relations between user action and sensitive Application Programming Interface (API) calls providing system functions. Based on this idea, they set up the desirable behavioural patterns database that can be used to recognise legitimate actions in programs. In contrast, if the software makes sensitive API calls without proper user triggers, and the behavioural patterns do not have any records in the desirable behavioural patterns database, then this application will be classified as malicious.

Dynamic analysis solutions [11,24,25] monitor an application's behaviour while the application is running. Mahindru et al. [11] chose 11,000 Android apps for analysis, focusing on the permissions the software needs to be installed and run. Combined

with the results of their study, they built a database to classify normal and malware behaviours. They tried different machine learning methods for detecting malicious Android applications. Unlike [11] who focused on app permissions, Zhou et al. [9] proposed a new dynamic analysis solution to detect malware apps in the Android environment. They suggested modifying the Android OS source code for indicating and recording app runtime system calls. In this way, they collected the behaviours of different apps and classified dynamic features of apps into a 166-dimension matrix. Then, they built a machine learning algorithm to detect malware apps. However, hackers can use obfuscation techniques to generate variants of an existing malware family for evading detection. Some dynamic analysis solutions require the modification of the existing source code for collecting specific events that are triggered by the program. However, doing this may result in breaking the app's integrity.

2.2 Malware detection using deep learning

With the maturity of deep learning technology and its successful applications in several fields, many researchers [15–17,26,27] have started to investigate how to apply deep learning techniques to detect malware. For instance, Nataraj et al. [15] first used image processing techniques to convert binaries into a gray-scale image. Then, they grouped similar malware into the same family and discovered the malware from the same family appears very similar in layout and texture. Inspired by their findings, Kalash et al. [16] and Venkatraman et al. [17] investigated different deep learning models for malware detection. Kalash et al. [16] used a Convolutional Neural Networks (CNN) approach for malware classification. Unlike existing approaches that rely on a specific training set to achieve high accuracy, the proposed solution can learn the discriminative representation from the input data. The proposed model can also automatically adjust to accommodate changes in input values. In contrast, Venkatraman et al. [17] focused on a computationally cost-effective and scalable hybrid solution by combining similarity mining (which is a machine learning approach to measure the similarities between two objects [28]) and deep learning solutions together. The proposed solution uses less computation resources when compared with existing machine learning solutions. Their proposed technique also resulted in high accuracy of 99% when detecting malware from the Microsoft Malware Classification Challenge (BIG, 2015) dataset [28]. The deep learning techniques bring intelligent approaches for detecting malware; however, the solutions assume that we have a good training set that contains known malware. Unfortunately, current solutions based on deep learning still cannot achieve 100% accuracy.

Although both approaches are implemented using different technologies, the core idea is to find similarities, then classify malware into different categories. However, the major drawback of both approaches is their inability to verify unknown malware.

2.3 Sandboxing and OS blocking technique against malware

To protect the system and programs from malware, sandboxing is used by software platforms. Sandboxing is an isolated environment to assess the application from critical

system resources and other programs. Hence, the system installs the software in a restricted area and monitors its behaviour. If the program violates a predefined policy, it will be deleted to prevent a possible attack. Some researchers [18,19,29–31] have started to investigate how to improve the efficiency and accuracy of the sandboxing approach. Researchers et al. [29–31] used sandboxing to extract system calls invoked by the well-known malware and then using those calls as the patterns to detect the malicious activities of the malware. To extend this idea, Wang et al. [18] use the machine learning approach to analyse the misbehaviour and inject the instrumentation code in the isolation environment. Although sandboxing solutions provide an isolated environment to assess and analyse threat behaviour, Vcisar et al. [19] discovered some drawbacks of using the sandboxing approach, such as hackers use different logic for programs running in a virtual machine environment to prevent their programs from being identified as malicious. Unlike sandboxing solutions, which create a separate isolated environment, Shan et al. [20] and Xing et al. [21] use OS protection to block malware installation. So, their solution configures a whitelisting that allows specific software to be installed. However, jailbreaking or pirated OS is the act of removing security restrictions of the OS imposed by the vendor.

2.4 Blockchain-based malware detection

In recent years, blockchain technology has matured significantly with its capabilities such as immutability, transparency, auditability, data encryption, and operational resilience. Several researchers [32–34] paid attention to the blockchain technology, hoping to use this technology to solve the problems in software management. For instance, Herbert et al. [33] propose a blockchain-based system for validating the software licence in a peer-2-peer distributed network. They introduced two software licence validation models: 1) Master Bitcoin Model and 2) Bespoke Model. The master bitcoin model uses a unique blockchain address to represent ownership of the software, and this ownership is updated after the master bitcoin is transferred from the current blockchain address to a new address. The Bespoke model utilises a token that represents an entitlement to a specific software application. So, a user who holds a particular token from a specific software vendor has a right to use the software. Yet, the proposed solution does not stop users to download malware. Du et al. [32] rely on software release platforms to audit software security. Once the software has been audited, the users can safely download it. However, the solution has some drawbacks. The first drawback is high cost, i.e., the software download platform needs to do penetration testing for each software. Nevertheless, this is too expensive for websites that provide free downloading because they do not make money by downloading software. The second issue is information transparency, i.e., if a user has been attacked by the software, it is difficult for other users to know this information. Unlike Du's solution, Homayoun et al. [34] presented a Blockchain-Based Malware Detection Framework (B2MDF) for discovering the malicious mobile applications from the app stores. The proposed solution is built by the private blockchains and a consortium blockchain. A Detection Engine (DE) uses the feature blocks which save in the Internal Private Blockchain (IPB) and the scanning information of different versions of applications

from the External Private Blockchain (EPB) to classify an app as malicious or benign. Each DE is defined as an active participant in the Consortium Blockchain (CB), only more than 2/3 participants agree that an application is malicious, then the DE can append a new block to the app's CB. Although the proposed scheme prevents users from downloading malware from app stores, there is a lack of a mechanism for users to report malware and how to reward users who identify malware.

To address this issue, we introduced security auditors in our solution and we use their professional knowledge to find the unknown threat and improve the detection accuracy. We also provide an open platform to allow victims and security experts to report malicious software and reward users who participate in this ecosystem.

3 Our methodology

In the past few decades, malware has been a hot topic. Hackers use malware to steal personal information [1], commit financial crimes [35], and target public infrastructures [36]. Therefore, it is important to make sure that all vulnerabilities present in Internet-connected devices are fixed. In this context, in this section, we present an ecosystem to manage software security.

3.1 System high-level design

As we mentioned earlier, the current process of software downloading and installation is too centralised and is not transparent. This can cause end users to install malware without fully assessing the security level of the software. In this study, we propose a software assessment ecosystem that brings in some features from blockchain technology, such as transparency, traceability and immutability. Such features help us to trace the release and modification information for each released software. Additionally, our proposed ecosystem can prove to our end users that the information in this system has not been tampered with. We also try to ensure that this system can trace any modification of the software and broadcast the changes to all users participating in the system. Figure 2 presents the main components of this new ecosystem which is composed of three blockchain networks: release blockchain, audit blockchain, and announcement blockchain networks. Furthermore, developers will provide software release information to blockchain networks. Users can provide information about known malware to blockchain networks, or check whether the software application is malware. The role of the security auditor is to assess the security level of the queried software from the blockchain networks. We describe below the main functionality of each entity in this ecosystem:

- **Users:** represent people who download or use the software application. Further, users are able to query or report malware.
- **Developers:** refer to a group of people who develop software based on user requirements.

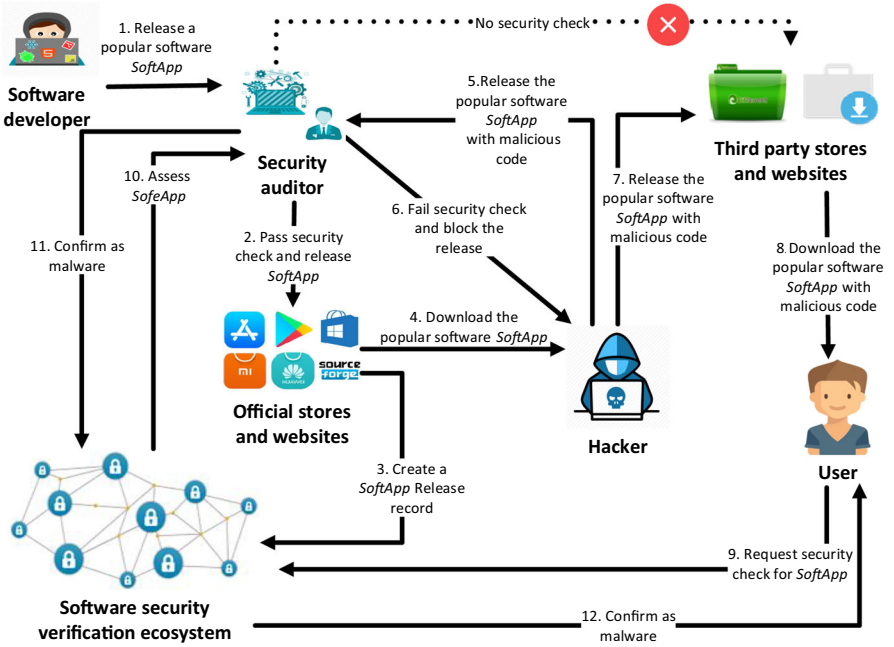

Fig. 2 Overview of our proposed approach in which users check the integrity information of each downloaded software application using the proposed ecosystem (Steps 1–9). The security auditors who participate in this ecosystem assess the user request and verify the security risks of the requested software in Step 10. Once more than 50% security auditors agree that this is malware (Step 11), the ecosystem will inform the user not to install *SoftApp* (Step 12)

- **Hackers:** are people with professional computer skills, who illegally obtain users' private information through some unusual means or obtain an OS privilege to access a PC without user permission.
- **Security auditors:** are people who have security knowledge and experience. They can use their expertise to determine whether the software application has security vulnerabilities or not. Moreover, security auditors are responsible for discovering the users who provide false alarms, i.e., users who claim the software has malware when in fact it does not.
- **Release blockchain networks:** provide the software release history, which can help the end users to verify the integrity of the downloaded software application. In this blockchain network, a block contains the version number of the software, the hash from the previous block, the timestamp when the block is generated, the digital signature, the uploader's information, the nonce and a difficulty target specifies the number of leading zeros. A developer generates a block by computing a hash with the nonce to meet the difficulty target. The computation process is called *proof of work* [37]. When a user downloads software from a third party website, the user can query the release blockchain network for pulling the release information of software. Next, the user can use the record from the blockchain network to

compare the released information from the downloaded software application, such as comparing the consistency of the hash.

- **Audit blockchain networks:** allow end users to report suspicious software or query the software security level. For malware reports, the primary task of security auditors is to verify the authenticity of the reports. For example, whether the same security vulnerabilities can be replicated by repeating the steps mentioned in the report. For those users who provide false information, we need to record their usernames or IDs to evaluate their credibility later. Hence, the malware report contains the software version, the description of the security incident, the timestamp when the block is generated, the digital signature along with who reported the incident and where did the user download the software. A difficulty target is used to indicate proof of work. In contrast, if a user queries the security level of a downloaded software application, security auditors need to assess the risks of the software in detail. This procedure may be done by launching a penetration test for detecting the software vulnerabilities, or by installing and running the software in a real environment to monitor if there is any abnormal behaviour. In the process of checking the security level of software, security auditors need to use a lot of tools and experience to prove their conclusions. So, we require users to specify the virtual credit points spent on this security verification. This virtual credit point will be used to reward auditors for completing the software security verification.
- **Announcement blockchain networks:** contain information about the malware. Once security auditors confirm that the security incident provided by end users is reliable and true, the ecosystem generates a block to record the reported malware. This block includes the software version, the description of the security incident, the timestamp when the block is generated, and the digital signature along with the place to download this software. Moreover, a difficulty target indicates proof of work along with the pseudonyms information of the whistleblower and the pseudonyms information of the first three security auditors who verified software. We discuss privacy concerns in Sect. 4.

3.2 Incentive mechanism

We introduce an incentive mechanism to reward those users who provide malicious reports or security auditors who evaluate software security risk. The main aspect of this mechanism is using virtual credit points to encourage participants to participate in this ecosystem. These virtual credit points can be used to represent the participants' credibility within the network as well as using these virtual credit points as a reward for asking the security auditors to verify the security level of a downloaded software application. For every registered user, developers, platforms and security auditors, the ecosystem issues initial virtual credit points. Users can use these virtual credit points as a reward to ask security auditors to verify the software security level. On the other hand, these initial virtual credit points are a baseline for representing the reputation of security auditors, platforms and developers. If a developer is found uploading or distributing malware, the ecosystem will deduct her virtual credit points. A user should not be able, or at least be warned, to download software from a website or store when

her virtual credit points fall below this baseline. Additionally, a website or store should not accept software from a developer who has lost the virtual credit points many times.
Rewards for security auditors: security auditors might receive the virtual credit points after they provide their analysis of the security incident in the audit blockchain network. For instance, a security auditor picks up a security incident and demonstrates how to replicate the same incident or highlight the vulnerabilities of using this software. Once everyone gives their analysis, and more than 50% of people have the same conclusion, then the people who gave their analysis first will receive the virtual credit points as a reward. These virtual credit points can be used to reflect their expertise. They can redeem these virtual credit points.
Rewards for users: end users can use some of these virtual credit points for seeking help from security auditors in order to determine the security level of the downloaded software, or to query the software release history. In contrast, end users can also earn the virtual credit points from the system. They can provide the malware information to the ecosystem, such as a user reports malware to the audit blockchain network, when the audit blockchain network confirms the malware. The user will receive virtual credit points as a reward that appreciates the user's contribution to the ecosystem.

3.3 Flow of malware detection

Figure 3 demonstrates the process of malware check. For detecting and announcing the malware in the ecosystem, we rely on the following events: user reports, user requests, auditor verification, and disclosure of information. In user report events, users can request a security inspection for the software that has an integrity issue, such as an inconsistent integrity value. Alternatively, users who have been attacked can report the malware that caused the attack. To generate a user report, first, a user queries the announcement blockchain network for checking the software security record. If there are no records, the second step checks the release information of the requested software by providing the software name, version, and the hash value. The software release information is created by the software developers after releasing new software. Such information will be used by the user to compare the integrity of the downloaded software. If there is any inconsistency between the downloaded software and the release information from the ecosystem, such as the hash value, the software version number, or the software name, the user can then generate the user report and submit it to the audit blockchain network for reporting a potential malware. On the other hand, if a user wants to check the security level of a downloaded software application, the user needs to pay some virtual credit points for requesting the software inspection. At the same time, a security auditor can earn some virtual credit points after verifying the security level of the required software. For an auditor verification event, we design a distributed voting mechanism based on the consensus protocol for distributing trust. Security auditors provide the analysis reports and then vote on whether the software is malware or not. After more than 50% of security auditors agree that the software is malware, the system will disclose the information event. The disclosure of the information event informs all users, websites, stores in the proposed ecosystem not to download or distribute the software. Besides, the software will be marked as malware

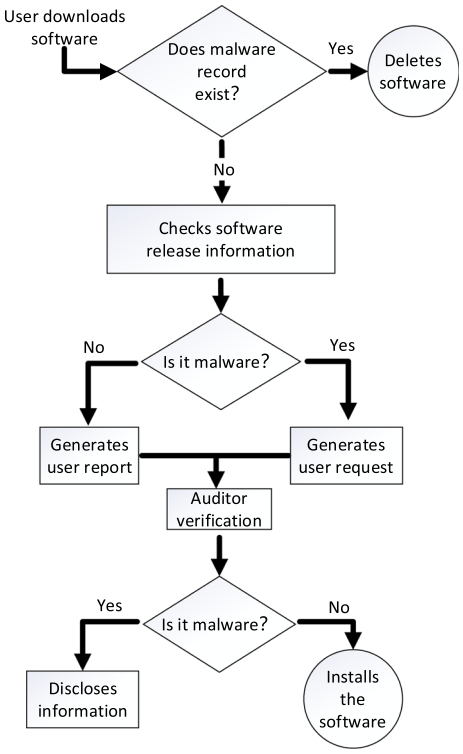

Fig. 3 The process of checking malware

and put into a block for facilitating the user to check later. Next, we describe each event in detail.

3.3.1 User reports

As we have mentioned previously, there are two user cases for generating a user report. In the first scenario, a user downloads software from a third party website. Next, the user finds the software has an integrity issue, such as the downloaded software application contains a hash value which is not recorded in the release blockchain network. In the second scenario, a user is attacked by the downloaded software application. In this case, the victim wants to warn other users not to download the same software as well as requesting websites and stores to delete this malware. Next, we describe both user scenarios in detail and explain how to generate a user report block in the ecosystem.

Scenario one: the integrity issue If a user downloads software from an unofficial website, how can the system help users determine if the software has been injected malicious codes? Can the user trace the historical release information of the software? Fig. 4 depicts how users can request the software release history from the proposed ecosystem, such as performing integrity check to determine whether the software has been modified or not. For example, the official stores and websites create a release record of a popular software *SoftApp* and then push it to the release blockchain network

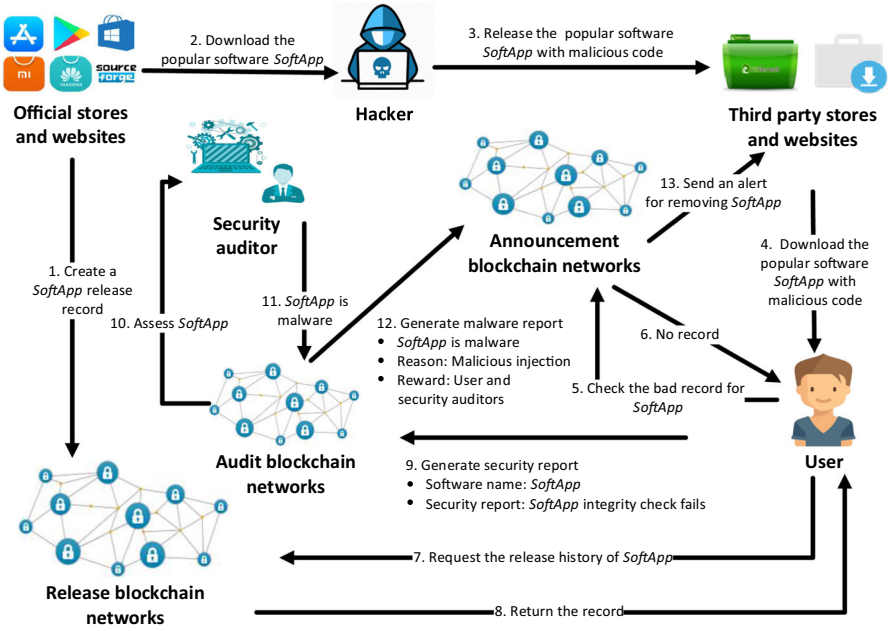

Fig. 4 How to use the historical release information to detect the malware injection attack

(Step 1). This record contains the version number of the software, the hash value, the timestamp when the block is generated, the digital signature, the information of the whistleblower and a difficulty target to indicate proof of work. Next, a hacker downloads *SoftApp* from the store (Step 2), modifies *SoftApp* by injecting malicious codes, and then pushes this modified *SoftApp* version to third party platforms (Step 3). An end user downloads the modified *SoftApp* from the third party website (Step 4) and queries the software security information from the announcement blockchain network (Step 5). If there is a record that indicates this is malware, the response will inform the user and warn the user not to install this software. Otherwise, the response will indicate that no record has been found (Step 6). Then, the user can send a software version to the release blockchain network (Step 7) for querying the software release information. The release blockchain network returns the record that contains the software version, the hash value and owner information etc.. (Step 8). The user generates a block if the release information is found to be inconsistent (say the hash value of the downloaded software is not consistent with the hash value stored in the blockchain network), the user creates a block to save this mismatch information and adds this block to the audit blockchain network. This block includes the software version number, the security incident report, the timestamp when the block is generated, the digital signature along with who reported the incident and the place to download this software as well as a difficulty target to indicate proof of work (Step 9). Security auditors pick up the user request and assess the security risk of *SoftApp* (Step 10). If more than 50% security auditors confirm that the *SoftApp* is malware (Step 11), then the ecosystem generates a new block to save this malware information in the announcement blockchain network

and reward the participants involved(Step 12). Finally, the ecosystem sends a warning alert to inform third party platforms to remove the malware (Step 13).

Scenario two: Download malware from untrusted platforms Figure 5 demonstrates the entire process of downloading malware from a third party website not trusted by our platform, which brings the virus to infect the user's PC. The user warns other users not to download and install the software through our platform. In this process, the user checks whether the downloaded software has bad records (Step 4). The system returns no record for this software (Step 5), but warns the user not to download the software from the untrusted platform. The user then checks the software integrity by querying the software release history from the release blockchain network. The software integrity check returns no record (Steps 6–7) and suggests the user not to download the software from the untrusted platform. Although our platform sends the warning message to the user, the user can choose to download software from the untrusted platform. However, the user finds that her computer is infected with a virus after installing the downloaded software application (Steps 8–9). As a result, the user wants to inform all users, websites in this ecosystem to stop using this software application. So, the user generates a new block (Step 10), which contains the software version, the security incident report, the timestamp when the block is generated, the digital signature along with who reported the incident and the place to download this software as well as a difficulty target to indicate proof of work. The user then sends this block to the audit blockchain network for security auditors to prove the security incident report in this block. After the audit process (Steps 11–12), the security

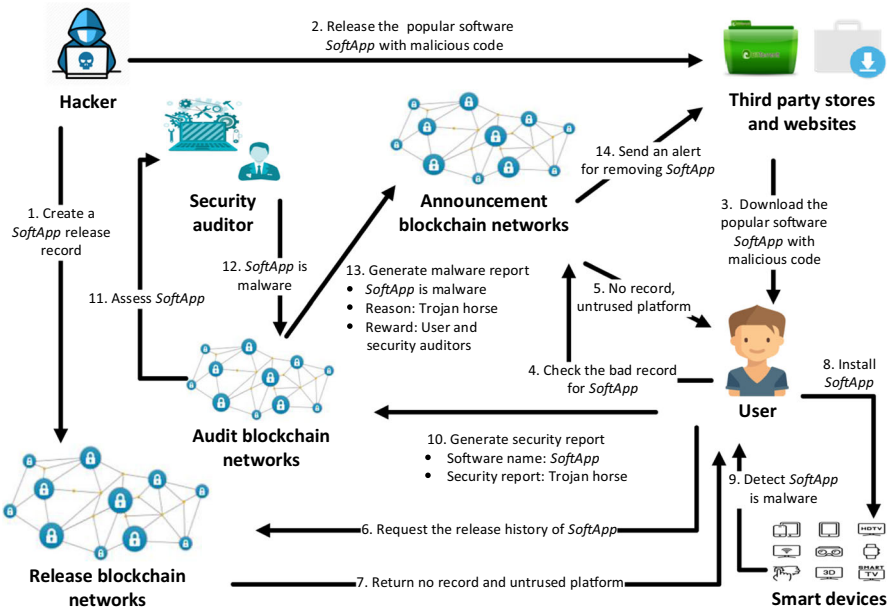

Fig. 5 The overall process of reporting malware after being attacked

auditors select the pending blocks from the audit blockchain network, then provide their conclusion. In this example, more than 50% security auditors think the *SoftApp* is malware. As a result, in Step 13, the announcement blockchain network receives a block that contains the software name, the security incident report along with the incentive information. Finally, the ecosystem informs all the third party websites or stores to remove this malware (Step 14).

3.3.2 User request

As in the example shown in Fig. 5, a user checked the software integrity and verified that the downloaded software application had no vulnerability record. Thus, the user installed the downloaded software application. However, the downloaded software application contains malicious code and harms the user's PC. To solve this issue, the user can generate a user request event block that contains the software version, the timestamp when the block is generated, the digital signature along with who has requested this check and the place to download this software as well as a difficulty target to indicate proof of work. Furthermore, the user needs to specify the virtual credit points spent on this security verification. As a result, the security auditors participating in the security verification will split the virtual credit points as their rewards.

3.3.3 Auditor verification

In the audit blockchain network, the security auditors verify the security level of the requested software and try to replicate the same issue that has been mentioned in the security incident report. If a security auditor confirms this is a malicious piece of software, the security auditor will vote 'Agree'; otherwise, the auditor will choose 'Disagree'. When more than 50% of people think this is malware, the report block will be pushed to an announcement blockchain network. This block contains the following information: the software information (software name, version and hash value), the security incident report, the timestamp when the block was generated, the person who issued this report; the first three people who verified and confirmed this issue, the source where users downloaded this software, the digital signature as well as a difficulty target to indicate proof of work. Taking the potentially vast amount of user reports into consideration may bring a considerable burden to security auditors to complete the process. To solve this issue, we can offload the requests by subdividing all security auditors into different assessment groups. Each group is allocated a certain number of requests. In this design, we can limit who can process the software assessment request. As a result, it is possible to enforce a policy that can minimise the burden. To ensure that we can achieve more than 50% voting, we can select an odd number of auditors from each group.

3.3.4 Disclosure of information

Once security auditors confirm that the software is malware, the announcement blockchain network will receive a new block. The ecosystem extracts the related information (such as the name of the software, the version number of the software, and the

place to download the software) from this block. After this information is obtained, the ecosystem system performs the following steps. First, it informs the software owner or providers to remove the malware immediately. Furthermore, the ecosystem deducts the virtual credit points for software providers, the websites, or stores that provided this malware. Second, for the first three security auditors who have participated in the software verification process, the ecosystem will send virtual credit points as a reward. Third, the malware information will be saved as a bad record for helping users not to download the software.

3.4 Implementation, deployment and maintenance

Based on the work of [38], we recommend that the existing software download platforms such as Apple Store, Google Play Store, or Chrome App Store to be part of this open-source platform. They can decide the standard of a security audit, choose the type of blockchain network, and discuss the information contained in each uploaded block. For the blockchain network implementation, many existing blockchain platforms have been widely used in different fields. We suggest users to use the existing blockchain network for uploading the results of the software security audit. The Hyperledger Fabric network [39] can be a potential candidate due to the high throughput and low latency. For the deployment, there might not be so much malware information in the blockchain network at an earlier stage. So, the auditors need to use some automated malware detection technologies [40,41] or databases [42] to reduce the auditing speed. In this proposed solution, we encourage the software downloading platform to (such as Google Play Store or Apple Store) assess the security level of the software before publishing it. This design ensures that users do not need to wait when downloading the software. From the perspective of maintenance, we recommend that the well-known platforms act as ledgers in the network to share the status of uploaded software and malware records. We believe that the proposed system should be maintained by regular users and the platform for system maintenance. For example, Google Play Protect [43] allows users to upload apps they do not know or receive from unknown resources. Google Play Protect then helps users review such apps' security. The purpose of using this feature is that users sometimes do not download apps from known or trusted platforms. So, the user wants to assess the software security. Therefore, when users upload such apps, the software platform helps them to evaluate the software's safety and informs other platforms to reject any kind of malware.

4 Security issues and performance challenges

The proposed solution aims at ensuring that users do not use or download malware from the Internet. Therefore, this solution is likely to be targeted by attackers, such as hackers can break system availability, destroy system credibility, and overload system performance. In this section, we discuss the possible security and performance issues that this ecosystem may face as well as providing the relevant solutions to address each security risk or performance issue.

DoS attacks In practice, we could deploy the ecosystem under a DoS resilient infrastructure, i.e., we could use a Content Delivery Network (CDN) or a cloud platform. These infrastructures have their built-in security mechanisms to mitigate Denial of Service (DoS) and Distributed DoS attacks; for instance, load balancers can redirect live traffic from one server to other servers if the current server becomes unavailable. An auto-scaling technology can be used to add additional resource to handle the increased server load. Also, all blocks in this ecosystem are either user-generated or generated by security auditors. The advantage of this design is to prevent hackers from attacking a third party entity that generates blocks.

Rogue user reports and requests In our proposed ecosystem, all the actions are recorded in the blockchain, which is visible for a public audit. In Fig. 3, the malware validation process includes two events: user reports and user requests. Every user report or request contains the reporter information. Once security auditors discover that a user tries to report a false alarm, the user who posted this report will be penalised, by reducing virtual credit points in the system. Inspired by [44–46], we introduce an intelligent system to remove entities who have malicious behaviours. This intelligent system will remove misbehaviour entities out of the ecosystem if the virtual credit points are much lower than the initial credit points. However, detecting and investigating a malicious user by using the virtual credit points is a complex and difficult process, because the user might spend all the virtual credit points in verifying the security of the software, which would lower her virtual credit points. To address this issue, we introduce a new parameter to record how many rogue reports the user has posted, as well as keeping track of the number of requests that the user had been sending to the audit blockchain network over a period of time. If one of the above conditions is met, then we put the user on the blacklist and remove her later.

Auditor verification overhead If users constantly make requests or reports to the audit blockchain network, this will cause many pending blocks to be processed. Based on this scenario, security auditors may not be able to deal with all the requests on time. As a result, it will increase the overhead on the auditor's verification process along with increased delays at the client side. To reduce the performance overheads, we can write some code to verify certain requests, such as performing the integrity check. Moreover, inspired by some recent studies [10,15,17], we can use deep learning technologies to detect malware. For instance, we can collect some malware samples from [15,47] as our training sets to build a learning model. Once the model is well trained, we will let the trained model analyse the software and tell us if there is a security risk. It is worth mentioning that the solutions mentioned above can only be used to reduce the performance overhead and detect some malware. The security auditors still play an important role to verify the security level of the requested software and provide reliable results.

Collusion attacks we propose a method of pre-configured trusted nodes to resist attacks from collusion groups. For example, antivirus software companies, certified software platforms, and software owners can be set as trusted nodes. The system will give higher trust and priority to their feedback. The auditors can be ordinary users, independent security engineers or antivirus software companies. Anyone can create an auditor. For this reason, we have required real-name authentication when a user registers as an auditor. Once these accounts are found to have launched a collusion

attack, we will block these accounts and its related real identities. So, they will no longer be trusted.

Privacy preservation When an auditor registers with the auditing ecosystem, we will assign a long-term certificate as its master certificate. At the same time, each auditor will also have a set of short-term certificates for using as a pseudonym. The master certificate contains the user's true identity and related information, which will be saved by the auditing ecosystem. The system periodically updates the short-term certificates. The auditor uses a pseudonym when sending the evaluation feedback of the software. The ecosystem will find its corresponding long-term identity certificate according to the pseudonym. As a result, the auditor will receive the reward. During the auditing process, the pseudonyms are constantly and effectively replaced in a timely manner, thereby protecting the auditor activities from being tracked. At the same time, we designed a mechanism to limit the time and frequency of using pseudonyms for preventing the Sybil attack.

5 Security and performance analysis

The proposed solution relies on victims to provide information about malware. Software vendors audit the software uploaded to the software platforms. Security experts conduct security tests of the software that will be uploaded to public software platforms. Therefore, we need to prevent hackers from impersonating ordinary users, or security experts from distributing malware. In this section, we briefly provide security analysis of our solution as well as discuss the impact of using different blockchain solutions on efficiency.

Security Analysis We assume that a hacker acts like a regular user and generates a false report, leading to the system making a wrong assessment, such as marking legitimate software as malware. For each user, the user has a 50% chance of being trustworthy and a 50% chance of being untrustworthy. Therefore, we use a binomial distribution to calculate the probability of untrusted users among all participants participating in the malware assessment. The binomial distribution is given by:

$$P(X = x) = \binom{n}{x} p^x (1 - p)^{n-x}$$

X denotes the number of untrusted users, n indicates the total number of participant, and p represents the probability that a user is untrustworthy. Figure 6 shows the probability of untrustworthy users among participants. By observing the trend of the graph, we found that the probability of 50% of users being untrustworthy is relatively high. Our results demonstrate that, as the number of users increases, the probability of users being untrustworthy gradually decreases. In our experiment, we chose 3 different samples, for example, 10 people participating in the malware assessment, and then we gradually increase the number of participants to 20 people and then 30 people. We found that, with this trend, the probability that more than half of the participants are untrustworthy is very low, is still maintained. However, based on the 51% rule in the blockchain network, it is challenging to control more than 51% of the comput-

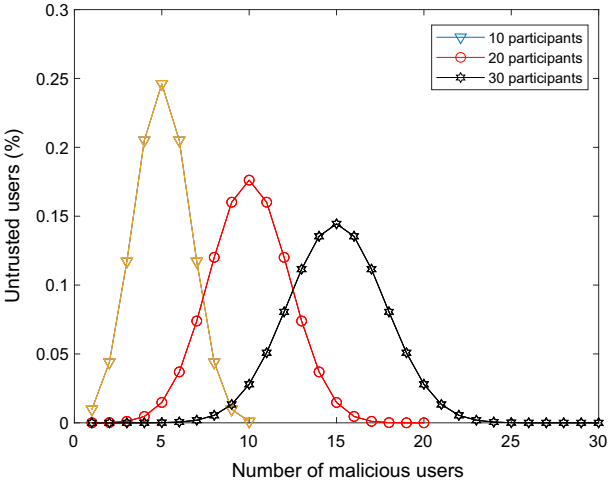

Fig. 6 Simulations of the probability of untrusted users among the number of security auditors

ing power within a blockchain network. Therefore, we believe if a hacker wants to control more than 50% of the participants in our network, the probability is very low. We introduce a credibility mechanism to the proposed scheme. If users and security auditors provide false information, their credibility will be reduced. When the system assesses the report provided by the participant, the system will accept the report based on the participant's credibility. If the participant's credibility is too low, the system will remove the participant from the existing blockchain network.

Performance Analysis Unlike the centralised solution, the blockchain must handle more tasks than the existing centralised solution, such as signature verification, consensus mechanisms, and redundancy. Therefore, we consider the performance when we use blockchain as a security solution. There are currently many blockchain platforms that take different times to generate blocks. For instance, Bitcoin [48] takes 10 minters to generate a block, while Ethereum [48] only spends 10 to 20 seconds on the block creation. In the past 5 years, many researchers have conducted studies on the performance of consensus, throughput, and system usage. For example, [49,50] highlighted that the throughput of the blockchain could reach 3534 transactions per second (tps) for Fabric, 533 tps for Libra and 40 tps for Ethereum. Different blockchain solutions have other effects on OS performance. In general, the usage of CPU and memory are also below 50% of the total resource. Therefore, users only need to choose the appropriate blockchain solution according to their needs to integrate with our solution. When considering our solution's deployment, we may configure some peers in different regions to hold the software status. Using such a design, we can minimise the latency by querying the software status from the closest peer. So, the synchronisation of peer's information may be affected by the network status, such as the packet loss or delay. Figure 7 shows how the number of peers or the available bandwidth affects the data synchronisation throughput.

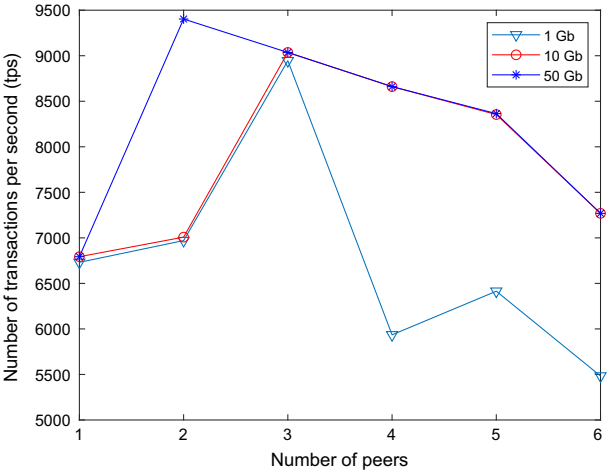

Fig. 7 Simulations of the throughput by increasing the number of distributed peers under different available bandwidths

Figure 7 shows that the number of peers has a certain impact on throughput. If the number of selected peers is too small, the throughput will not be sufficient when synchronising data, which will cause the data to not arrive in time. On the contrary, if the number of selected peers is too large, it will cause data loss in the synchronisation process and cause a delay. Also, we tried to observe the throughput under different available bandwidths, and we found that as the available bandwidth increases, the throughput also increases. We cannot estimate the performance of the malware software assessment process, it depends on the experience of security auditors and the difficulty of software evaluation. For example, to assess well-known malware, we assume that a security auditor needs 5 min to generate the assessment report and upload it to the proposed system. Then, the proposed system takes 1 minute to evaluate the credibility of the security auditors and aggregate the received results. Next, it spends 10–20 s to generate the block and upload it to the blockchain. In this case, the proposed system takes about 7–8 min to synchronise the software state to the blockchain network. However, for unknown issues, the assessment time may increase depending on the experience of the security auditors.

6 Conclusion and future directions

To verify the security level of the downloaded software, we have proposed a public malware verification ecosystem based on the blockchain protocol. We combine the characteristics of blockchain to provide a decentralised, transparency and tamper-proof software audit management. The outcomes of this proposed solution are: First, we design a distributed voting process based on the consensus protocol to avoid centralisation in practice; second, we proposed a reward and incentive mechanism, which makes sure all security auditors and users who participate in the system can con-

tinue maintaining the system; third, we designed different blockchain networks for forwarding traceability, public audit and security checking.

In the future, we plan to extend this work along several directions. To this end, we list down four directions. First, we aim to consider how to add Artificial Intelligence (AI) models to the proposed solution—using AI model to reduce the auditing overload from security auditors. Second, we have to consider the performance costs – such as Central Processing Unit (CPU) usage, memory usage, and battery consumption—in generating a block on mobile devices. Third, we will study how to reduce the computing cost in the blockchain networks while ensuring the security of the framework. Finally, we intend to investigate a monitoring mechanism to make sure that the announced malware is no longer available.

Acknowledgements We thank the anonymous reviewers for their valuable comments, which helped us improve the content, organisation, and presentation of this work.

References

1. McGee MK (2015) FBI alerts hospital to malware incident. https://www.databreachtoday.com/fbi-alerts-hospital-to-malware-incident-a-8710
2. Suryani V, Sulistyo S, Widyawan W (2017) Internet of Things (IoT) framework for granting trust among objects. J Inf Process Syst 13(6)
3. Cimpanu C (2017) HummingBad Android malware found in 20 Google Play Store apps. https://www.bleepingcomputer.com/news/security/hummingbad-android-malware-found-in-20-google-play-store-apps/#:~:text=Catalin%20Cimpanu&text=HummingBad%2C%20an%20Android%20malware%20estimated,Store%2C%20passing%20Google's%20security%20checks
4. Lakeshmanan R (2019) 'Agent Smith' malware replaces legit android apps with fake ones on 25 million devices. https://thenextweb.com/news/agent-smith-malware-replaces-legit-android-apps-with-fake-ones-on-25-million-devices
5. Cimpanu C (2018) Malware found in arch linux aur package repository. https://www.bleepingcomputer.com/news/security/malware-found-in-arch-linux-aur-package-repository/#:~:text=Catalin%20Cimpanu&text=Malware%20has%20been%20discovered%20in,intervention%20of%20the%20AUR%20team
6. SophosLabs, "who's your verisign?"—malware faking digital signatures (2010). https://nakedsecurity.sophos.com/2010/06/23/trojbhoqp-verisign
7. Li D, Li Q (2020) Adversarial deep ensemble: evasion attacks and defenses for malware detection. IEEE Trans Inf Forensics Secur 15:3886–3900
8. Chen Y, Shan Z, Liu F, Liang G, Zhao B, Li X, Qiao M (2019) A gene-inspired malware detection approach. In: Journal of physics: conference series, vol. 1168. IOP Publishing, p 062004
9. Zhou Q, Feng F, Shen Z, Zhou R, Hsieh M-Y, Li K-C (2019) A novel approach for mobile malware classification and detection in Android systems. Multimed Tools Appl 78(3):3529–3552
10. Elish KO, Shu X, Yao DD, Ryder BG, Jiang X (2015) Profiling user-trigger dependence for Android malware detection. Comput Secur 49:255–273
11. Mahindru A, Singh P (2017) Dynamic permissions based Android malware detection using machine learning techniques. In: Proceedings of the 10th innovations in software engineering conference. ACM, pp 202–210
12. Isohara T, Takemori K, Kubota A (2011) Kernel-based behavior analysis for Android malware detection. In: 2011 Seventh international conference on computational intelligence and security. IEEE, pp 1011–1015
13. Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Ribagorda A (2013) Evolution, detection and analysis of malware for smart devices. IEEE Commun Surv Tutor 16(2):961–987
14. Barrera D, Kayacik HG, Van Oorschot PC, Somayaji A (2010) A methodology for empirical analysis of permission-based security models and its application to Android. In: Proceedings of the 17th ACM conference on Computer and communications security. ACM, pp 73–84

15. Nataraj L, Karthikeyan S, Jacob G, Manjunath B (2011) Malware images: visualization and automatic classification. In: Proceedings of the 8th international symposium on visualization for cyber security. ACM, p 4
16. Kalash M, Rochan M, Mohammed N, Bruce ND, Wang Y, Iqbal F (2018) Malware classification with deep convolutional neural networks. In: 2018 9th IFIP international conference on new technologies. Mobility and security (NTMS). IEEE, pp 1–5
17. Venkatraman S, Alazab M, Vinayakumar R (2019) A hybrid deep learning image-based analysis for effective malware detection. J Inf Secur Appl 47:377–389
18. Wang C, Ding J, Guo T, Cui B (2017) A malware detection method based on sandbox, binary instrumentation and multidimensional feature extraction. In: International conference on broadband and wireless computing. Springer, Communication and Applications, pp 427–438
19. Čisar P, Joksimović D (2019) Heuristic scanning and sandbox approach in malware detection. Archibald Reiss Days 9(2)
20. Shan Z, Wang X, Chiueh T (2012) Enforcing mandatory access control in commodity os to disable malware. IEEE Trans Dependable Secure Comput 9(4):541–555
21. Xing L, Pan X, Wang R, Yuan K, Wang X (2014) Upgrading your Android, elevating my malware: privilege escalation through mobile OS updating. In: 2014 IEEE symposium on security and privacy. IEEE, pp 393–408
22. Drew J, Moore T, Hahsler M (2016) Polymorphic malware detection using sequence classification methods. In: 2016 IEEE security and privacy workshops (SPW). IEEE, pp 81–87
23. Drew J, Hahsler M, Moore T (2017) Polymorphic malware detection using sequence classification methods and ensembles. EURASIP J Inf Secur 1:2
24. Alzaylaee MK, Yerima SY, Sezer S (2017) Emulator vs real phone: Android malware detection using machine learning. In: Proceedings of the 3rd ACM on international workshop on security and privacy analytics. ACM, pp 65–72
25. Chin E, Felt AP, Greenwood K, Wagner D (2011) Analyzing inter-application communication in Android. In: Proceedings of the 9th international conference on mobile systems, applications, and services. ACM, pp 239–252
26. Yuxin D, Siyi Z (2019) Malware detection based on deep learning algorithm. Neural Comput Appl 31(2):461–472
27. Demetrio L, Biggio B, Lagorio G, Roli F, Armando A, Explaining vulnerabilities of deep learning to adversarial malware binaries. arXiv:1901.03583
28. Srivastava V, Biswas B (2018) Mining on the basis of similarity in graph and image data. In: International conference on advanced informatics for computing research. Springer, pp 193–203
29. Ali M, Shiaeles S, Papadaki M, Ghita BV (2018) Agent-based vs agent-less sandbox for dynamic behavioral analysis. In: 2018 Global information infrastructure and networking symposium (GIIS). IEEE, pp 1–5
30. Jain S, Choudhury T, Kumar V, Kumar P (2018) Detecting malware and analysing using sandbox evasion. In: 2018 International conference on communication, computing and internet of things (IC3IoT). IEEE, pp 111–116
31. Darshan SS, Kumara MA, Jaidhar C (2016) Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm. In: 2016 11th international conference on industrial and information systems (ICIIS). IEEE, pp 534–539
32. Du Y, Liu C, Su Z (2019) Detection and suppression of malware based on consortium blockchain. In: IOP conference series: materials science and engineering, vol. 490. IOP Publishing, p 042031
33. Herbert J, Litchfield A (2015) A novel method for decentralised peer-to-peer software license validation using cryptocurrency blockchain technology. In: Proceedings of the 38th Australasian computer science conference (ACSC 2015), vol. 27. p 30
34. Homayoun S, Dehghantanha A, Parizi RM, Choo K-KR (2019) A blockchain-based framework for detecting malicious mobile applications in app stores. In: 2019 IEEE Canadian conference of electrical and computer engineering (CCECE). IEEE, pp 1–4
35. Doffman Z (2019) Cybercrime: 25% of all malware targets financial services, credit card fraud up 200% . https://www.forbes.com/sites/zakdoffman/2019/04/29/new-cyber-report-25-of-all-malware-hits-financial-services-card-fraud-up-200/#47ec59807a47
36. Ashford W (2013) Malware hits US power plants. https://www.computerweekly.com/news/2240176164/Two-US-power-plants-hit-by-malware

37. Kumar G, Saha R, Rai MK, Thomas R, Kim T-H (2019) Proof-of-work consensus approach in blockchain technology for cloud and fog computing using maximization-factorization statistics. IEEE Internet Things J 6(4):6835–6842
38. Laurie B (2014) Certificate transparency. Commun ACM 57(10):40–46
39. Androulaki E, Barger A, Bortnikov V, Cachin C, Christidis K, De Caro A, Enyeart D, Ferris C, Laventman G, Manevich Y et al (2018) Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp 1–15
40. Xiao G, Li J, Chen Y, Li K (2020) Malfcs: an effective malware classification framework with automated feature extraction based on deep convolutional neural networks. J Parallel Distrib Comput 141:49–58
41. Agrawal P, Trivedi B (2020) Automating the process of browsing and downloading apk files as a prerequisite for the malware detection process. Int J Emerg Trends Technol Comput Sci (IJETTCS) 9(2):013–017
42. AV-test, Av test the idndependent it-security insitute (2020). https://www.av-test.org/en/
43. Cunningham E (2017) Keeping you safe with google play protect. https://support.google.com/googleplay/answer/2812853?hl=en
44. Spathoulas G, Collen A, Pandey P, Nijdam NA, Katsikas S, Kouzinopoulos CS, Moussa MB, Giannoutakis KM, Votis K, Tzovaras D (2018) Towards reliable integrity in blacklisting: facing malicious IPs in ghost smart contracts. In: 2018 innovations in intelligent systems and applications (INISTA). IEEE, pp 1–8
45. Huang C, Wang Z, Chen H, Hu Q, Zhang Q, Wang W, Guan X Repchain: a reputation based secure, fast and high incentive blockchain system via sharding. arXiv:1901.05741
46. Pereira S, Satish S (2013) Communication-based host reputation system, US Patent 8,381,289
47. Microsoft, Microsoft malware classification challenge (big 2015) (2018). https://www.kaggle.com/c/malware-classification
48. Siriwardena P (2017) The mystery behind block time. https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a
49. Dabbagh M, Choo K-KR, Beheshti A, Tahir M, Safa NS (2021) A survey of empirical performance evaluation of permissioned blockchain platforms: challenges and opportunities. Comput Secur 100:102078
50. Zhang J, Gao J, Wu Z, Yan W, Wo Q, Li Q, Chen Z (2019) Performance analysis of the libra blockchain: an experimental study. In: 2019 2nd International conference on hot information-centric networking (HotICN). IEEE, pp 77–83