SMARTWITNESS FOR SOFTWARE PACKAGE DISTRIBUTIONS

Field of the Invention

[0001] The present invention relates to software distribution. More specifically, the present invention relates to a system and method for distributing software with smart-witness capability.

Background

[0002] Package (software) managers, have become an essential part of solutions for software distribution and management. The main function is to allow users to install programs, drivers, or updates in their systems in a secure, quick, and often, unattended way. Overtime, attackers discover flaws in software distribution systems. Further, actors of the software supply-chain are generally ineffective in detecting and stopping attacks on user devices' end.

[0003] Package managers are now adopted in all the major operating system (OS) and platforms, such as APT in GNU/Linux, Play Store in Android, PyPi for Python programming language, and as mechanisms of firmware update for IoT devices. Behind the scenes, these systems rely on widely deployed public-key infrastructures (PKIs) to securely provide authenticated software packages. In a software supply chain, there may involve stakeholders and actors in the distribution process. Due to the complexity and limitations of distribution process, users may end up installing software malware or vulnerable software on their devices, even though they gotten it from the package manager. Despites existing countermeasures available on the package managers,

attackers continuously discovered ways to compromise user devices to distribute malware.

[0004] For instance, it was widely reported that, in March 2019, ASUS' software update system was compromised and used to spread signed malware over one million devices.

[0005] It was observed that those attacks took place due to multiple reasons. Firstly, most of the actors' actions, such as issuing a code signing certificate or releasing a software package, are not visible and accountable for public scrutiny. Secondly, the security services (e.g., antivirus manufacturers) play an essential role in identifying and stopping malicious and vulnerable software distribution, usually at the late part of the software distributions, whereby malware or vulnerable software may have already been spread. Lastly not least, users are often unaware of such security risks. Many of them do not subscribe, buy, or use security products. It was even surprising that users using the security software with even a critical vulnerability are not aware of this fact until the patch is released and applied (even if the vulnerability is publicly known).

[0006] As shown in FIG. 1, which illustrates a known code signing certificate revocation process, which is based on Public-key Infrastructure (PKI) certificate signing for a trusted digital identity for implementing on the software distribution system. A Certification Authority (CA) issues a certificate to a Developer (Dev) whereby revocation is defined therein. With the certificate issued, the Dev releases a software package to a Distributor (Dist). Users may then pull the package from the Dist onto their own device and install the package on the devices. If the user device has any devise security protection software installed thereon, the protection software may check the

package for any security threat. During the checking, the protection software may obtain the binary (of the downloaded package) and send to a corresponding Security Provider (Sec Prov) and package analysis. Should there be any security threat found, The Sec Prov notifies the findings with the CA to revoke the certificate and to halt the package installation. It is known that the time to define the revocation data of code signing certificate however can take up to 6 months on average, which is not viable in the fast-moving trend of the software distributions.

[0007] It is therefore desired to create a secure, dynamic, proactive, and transparent software distribution system.

Summary

[0008] In one aspect of the present invention, there is provided a blockchain-based software distribution method for distributing a software package of a developer through a distributor, wherein a user of the software package obtains the same from the distributor through a package manager on the user's device, and a security provider accessible to the software package for security assessment. The method comprises receiving a request from the developer to obtain a witness contract for the software package from a Certificate Authority (CA), the request includes developer's account for declaring ownership over the software package; obtaining evaluation from the CA a legitimacy and identity of the request through out-of-band procedures; receiving the witness contract pertaining to the software package from the CA; registering a hash value for the witness contract by making a call through a transaction which is mined on a blockchain; receiving a security score of the software package based on a security assessment done by the security provider; publishing the software package along with a

verification proof, *proof*, wherein the proof comprises a block ID, proofs of the software package's registration, the developer's ownership, the security score and an expiry of the software package; distributing the proof that is verified by obtaining a Markle tree proof to one or more databases in the blockchain.

[0009] In one embodiment, the method further comprises periodically receiving updated security score from the security provider of the software package.

[0010] In another embodiment, the method further comprises sending a copy of the software package to the security provided for security assessment to render the security score.

[0011] In a further embodiment, the security provider periodically provides security assessments to the software package and scores the results thereof, regardless of any changes to the software package by the developer.

[0012] In yet another embodiment, all versions of the software package from the developer and the respective security scores are recorded on the witness contract.

Brief Description of the Drawings

[0013] This invention will be described by way of non-limiting embodiments of the present invention, with reference to the accompanying drawings, in which:

[0014] FIG.1 illustrates a known code signing certificate revocation process;

[0015] FIG. 2A illustrates a schematic diagram of a blockchain-based software distribution system in accordance with an embodiment of the present invention;

[0016] FIG. 2B illustrates a flowchart of the blockchain-based software distribution system in accordance with an embodiment of the present invention;

[0017] FIG. 3 exemplifies a witness pseudo-code in one embodiment of the present invention;

[0018] FIG. 4 illustrates a SmartWitness storage in accordance with an embodiment of the present invention;

[0019] FIG. 5 illustrates a SmartWitness Skeleton in one embodiment of the present invention;

[0020] FIG. 6A exemplifies a software package description of an embodiment of the present invention; and

[0021] FIG. 6B exemplifies a software package's proof of an embodiment of the present invention.

Detailed Description

[0022] In line with the above summary, the following description of a number of specific and alternative embodiments are provided to understand the inventive features of the present invention. It shall be apparent to one skilled in the art, however that this invention may be practiced without such specific details. Some of the details may not be described at length so as not to obscure the invention. For ease of reference, common reference numerals will be used throughout the figures when referring to the same or similar features common to the figures.

[0023] In one embodiment, there is provided a software distribution system based on distributed ledgers, herein also named as SmartWitness. It is desired that the system provides properties of binary transparency, useful and granular package revocation, and dynamic and proactive security assessment improving risk awareness for end-users. Users are able to view versions, security level, etc. of the published package before downloading and installing on their devices. All these new features are included within software distribution systems supported by existing infrastructures and there is no external security tool or installation on users' devices. SmartWitness keeps all actors and their actions accountable, and it enables security providers to participate earlier (i.e. prior to publication) in the software distribution process, directly influencing package installations on user devices. This is implemented by replacing the current code signing digital certificate by a blockchain-based smart contract on the software distribution system. It allows the system to publicly record and track actions of all actors involved in the software distribution systems. Developers digitally seal software packages by registering them in a distributed ledger, herein referred to as "SmartWitness". It further includes a rating module allowing security providers to publicly and dynamically score

the security of software packages based on findings and analysis from out-of-band tastings and evaluations. In one embodiment, the rating module also enables security providers to singly revoke sealed packages and stop their distribution not having to revoke a complete SmartWitness to prevent malware spread. Users can verify that a developer had publicly and transparently released a particular software package, and is aware of any security issue or vulnerability without requiring external software. The present system provides an improved transparency and attestation over the SmartWitness on software distribution systems. The system includes a blockchain or distributed ledger that proactively involve security providers in the software distribution chain.

[0024] The following will show details on how SmartWitness is integrated into an existing package manager system.

[0025] FIG. 2A illustrates a schematic diagram of a blockchain-based software distribution system **200** in accordance with an embodiment of the present invention. FIG. 2B illustrates a flowchart for a blockchain-based software distributions in accordance with an embodiment of the present invention. The blockchain-based software distribution system **200** is also referred herein as "SmartWitness" system. Under Markle Tree Proof (MTP) properties, which is implemented herein, the SmartWitness may contain data within itself and any stored information can be verified off-line. In one embodiment, Merkle Patricial Tree is implemented when Ethereum platform is adopted for SmartWitness implementation. With the SmartWitness that include a programming logic, the software distribution system **200** of the present invention is more dynamic over the known static code signing certificate system, i.e. PKI certifications.

[0026] A developer (Dev) **202** initiates a process by requesting, at step **252**, a witness contract or simply "SmartWitness" **205** from a certification authority (CA) **204**. The request (for witness contract) includes a developer's account address for declaring ownership in the witness contract. In the present embodiment, the CA **204** serves as a central party bootstrapping the SmartWitness process to record all the activities onto the blockchain. In an alternative embodiment, a dedicated SmartWitness server may be used.

[0027] The CA **204** evaluates the developer's request, legitimacy, and identity through out-of-band procedures. If all provided data in the request is correct, the CA **204** proceed to issue, at step **254**, a new witness contact **205** and grant the ownership to the Dev **202**. In one embodiment, issuance of the new witness contract **205** is shown by in the pseudo-code in FIG. 3A. Then, the CA **204** notifies the Dev **202** about the issued witness contact **205** in response to the request. In one example, the witness contract's address is represented by:

[0028] $$witness_{addr} \leftarrow H(ca_{addr} \parallel ca_{nonce})$$

[0029] Before the Dev **202** releases a new software package for distributions through a distribution source (Dist) **206**, the Dev **202** registers, at step **256**, the new software package to be published. The registrations is performed by registering a hash value ($hash_{bin}$) to the new software package having the issued witness contract **205**. This is done by making a call through a transaction which is mined through a blockchain system and included in a block (*bi*). The new software package is then, at step **258**, released to the Dist **206**. The new software package contains a software binary file (*bins*) and metadata such as developer's public key (dev_{pk}), CA account address (ca_{addr}), CA

nonce (ca_{nonce}) at the time of witness contract's creation, and a signature (*sig*). The parameters can be represented as follows:

[0030] $M \leftarrow \{hash_{bin}, dev_{pk}, ca_{addr}, ca_{nonce}\}$

[0031] $sig \leftarrow \text{sign}(dev_{pk}, M)$

[0032] $Pkg \leftarrow \{bins, M, sig\}$

[0033] Considers that the transparent nature of the present invention, a security provider (Sec Prov) **208** constantly monitor all the actions performed by both the CA **204** and the Dev **202**. Again, as the SmartWitness is now available in the blockchain, the Sec Prov **208** does not need to constantly connect with the user device. That brings Sec Pro **208** to involve in the software distribution process in an early stage of software distributions, even before the package is pull onto the user's device **212**.

[0034] In one embodiment, when the Dev **202** registers a new package, the Sec Prov **208** obtains, at step **262** a copy, and/or the source code, of the software package to examine and make security assessments to the packages. Any conclusions or findings, e.g., vulnerabilities, Common Vulnerabilities and Exposures (CVEs) are registered in the blockchain in a form of a security score, at step **264**. At step **266**, the security score is recorded at the SmartWitness. To do so, security providers make a call to the corresponding developer's SmartWitness. In one embodiment, the call is a REST API call using Web3 APIs to instantiate the SmartWitness..

[0035] On the other hand, when the Dist **206** receives or pulls new software packages to-be-distributed from the Dev **202**, the Dist **206** checks an existence of the registration in the blockchain by the Dev **202** before publishing it. Then, the Dist **206**

attaches, at step **268**, a verification proof, denoted as *proof*, together with the security score evaluated by the Sec Prov **208**. Accordingly, the package would contain information as following:

[0036] $Pkg \leftarrow \{bins, M, sig, score, proof\}$

[0037] The blockchain platform, such as Ethereum, organizes properties and the storage of SmartWitnesses in MPTs, the software package's proof may contain the following data, such as the block number from what it was extracted (id_{block}), a proof of package registration ($proof_{bin}$), a proof of the developer's ownership over the SmartWitness ($proof_{own}$), proof of security score associated with the corresponding package ($proof_{score}$), and a proof that indicates the SmartWitness has not expired ($proof_{exp}$).

[0038] $proof \leftarrow \{id_{block}, proof_{bin,own,score,latest,exp}\}$

[0039] The *proof* is not required to be authenticated through any digital signature as the blockchain itself guarantees the authenticity. It only requires the distributor to repeats the aforesaid process to obtain and attached MPT's proofs to the packages.

[0040] When user **212** intends to install a package through the package manager, the package manager obtain the proof of that package at step **270** and published the same thereon. The package manager, at step **272**, triggers a package installation process by pulling a copy of the desired package and metadata thereof at step **272**. The package manager then performs multiple validations to ensure the provenance, authenticity, and integrity of the software package through the blockchain, at step **274**. The validations

rely on the user's copy of the blockchain headers (B_h) and a pre-defined list of trusted CAs (C). After validating the corresponding signature, the package manager checks if:

[0041] i) the CA is a trusted one: $ca_{addr} \in C$,

[0042] ii) the SmartWitness was indeed created by the corre- sponding CA*:*

[0043] $H\,(ca_{addr}||ca_{nonce}) = witness_{addr}$,and

[0044] iii) the validity of the SmartWitness, developer's ownership, software package registration and security,

[0045] $\forall b_h^i \in B_h \wedge i = id_{block} \Rightarrow \{root_{state}^i\} \subset b_h^i$:

[0046] $root_{state}^i = \gamma(proof)$

[0047] Wherein the block number, i, is a sequential number that works as an identifier, and γ represented a function of recalculating a root value from a provided MPT proof.

[0048] FIG. 4 illustrates a SmartWitness storage in accordance with an embodiment of the present invention. The SmartWitness storage consists of details of a package that are stored by the SmartWitness onto the blockchain. The SmartWitness storage comprises a witness contract, which include information such as, owner, expiration (date), ca_{addr}, packages information. Further, the packages information may include the $name_{pkg}$, a latest hash value, $hash_{latest}$, and all versions' hash value, $hash_{verx}$ and respective scores. When a software package is ready to be released, a developer registers a packages hash value ($hash_{bin}$) into the witness contract corresponds to the software package. It includes a software name and a version thereof. To ensure that the

package is authentic, the function *registerPackage*, can only be called by the owner of the package. According to the present invention, neither CA nor any third party can register software packages on behalf of the developer. The witness contract keeps a list of packages using a dictionary (denoted by *packages*). A key corresponds to a software name (e.g., “telegram-desktop”), and the value is a data structure that stores the hash value of the last version registered and using a second internal dictionary. This second dictionary denoted as *all_version* uses a hash of the package version as the key and an integer as its value, which indicates the security score.

[0049] FIG. 5 illustrates a SmartWitness Skeleton in one embodiment of the present invention. The SmartWitness Skeleton provides a basic function forming the SmartWitness for distributing packages. The security providers are positioned with a role to monitor the integrity of software package security by integrating a rating module. All packages, release by the developer or any third party, are assessed and graded by the security providers prior to publishing of the packages. According to the embodiment of the present invention, all packages published on the package managers require registrations into the SmartWitness, allowing security provider to track and be informed of any release. In one further embodiment, security providers may demand copies of packages for performing security tests and analysis. The findings from the security providers are scored between 0 (lowest) to 10 (highest) security level (of trust) of the package-to-be published. In yet another embodiment, only the security provider previously registered are allowed to score software packages. In one embodiment, the security provider is appointed by the CA. It can be appointed during the SmartWitness instantiation.

[0050] A SmartWitness is issued to a specific Dev's address by the CA. CA further has access to the SmartWitness to add security provider's address therein, which in turn allowing the security provider to update the security score periodically.

[0051] In case when the security provider detects signed malware, the rating module allows the security provider to revoke a specific software package with rating below a score threshold. Since distributors have to update package proof every k-blocks for publishing the package, the malware revocation would be effective once the distributors enable the new proof version ($proof_{i+k}$). This action provides the advantage of stopping a signed malware to be downloaded or executed on devices even though a full process of revoking the associated SmartWitness has not been done by the CA as shown in FIG. 2. Others legitimate software packages sealed using the same SmartWitness will not be affected.

[0052] The rating module of the security providers scores and indicates the security findings and outcomes based on their assessments of the packages. In one embodiment, the security providers may provide specific details, such as any vulnerability that has been found indicating the assigned CVE or, a developer refused to provide the software package for analysis, or other findings. This additional information may be added as finding parameters through a function *ratePackage*. While this additional information is not stored in the SmartWitness, it can be recorded in the blockchain by making use of logs. This allows the CAs to track security provider's actions and revoke any rating access in case of malicious activities originated from the security provider.

[0053] It is well known in the art that developers outsource distributions of their software using a Content Delivery Network (CDN) or mirroring architecture. With embodiments of the present invention, a more active role is assigned to the distributor to ensure the integrity of the distributed software packages. The distributor is responsible for attaching or embedding verification proofs to the package. The distributor then extracts proofs for the packages through MPT. The proofs may include at least the following information:

[0054] - a certain package is registered in a SmartWitness ($proof_{bin}$);

[0055] - package developer is the owner of that specific SmartWitness ($proof_{own}$);

[0056] - the last security score value assigned to the corresponding package ($proof_{score}$).

[0057] Moreover, any extracted proof is linked to the last mined block known by the distributor (i.e. $proof_x^i$). To guarantee up-to-dateness of such proofs, distributor extracts and re-attaches proofs of packages periodically, every k blocks (i.e. $proof_x^{i+k}$), such that the latest state version of the corresponding SmartWitness is available thereon. In another embodiment, even when there is no new updates or changes that require changes to the SmartWitness and/or packages' security score from the developers' side, the distributor continues updating the proofs over the blockchain. In the Ethereum platform for example, the MTP proofs are extracted by calling the JSON-RPC function *eth_getProof*, and it does not require to perform transactions; thus, it does not consume fees.

[0058] On the users' side, they are provided with the relevant information concerning the integrity of the package prior to the download. The distributors are the gateway to validate that the newly provided information, such as latest security score, is most updated and genuine. The validation ensures that a latest block known by the user and a block used to extract the proof should not be larger than a prescribed number of blocks, *k*, which otherwise, the package may be considered unauthorized.

[0059] On the package manager side, it validates the proof according to the MTP properties. A corresponding *stateRoot* is extracted from a specific block header of the proof. The integrity and authenticity of the proof are guaranteed by the supported blockchain. Once the obtain *stateRoot* value is match with the proof, user is ensured that the corresponding package to be installed is publicly registered in a SmartWitness issued by a trusted CA and the ownership is delegated to the developer who signed the package. Further, the integrity of the package is further ensured by the current value of the security score. In one embodiment, user may define a minimum-security score. When integrity of the package to be downloaded fits within the validation and rules set by the user, the package manager installed the required package on the user's device.

[0060] FIG. 6A exemplifies a software package description of an embodiment the present invention. The description includes developer's public key (Dev-PK), CA factory address (CA), and CA factory nonce (CA-Nonce).

[0061] FIG. 6B exemplifies a software package's proof of an embodiment of the present invention. The proof comprises the relevant information along with the corresponding blockchain address.

[0062] While specific embodiments have been described and illustrated, it is understood that many changes, modifications, variations and combinations thereof could be made to the present invention without departing from the scope of the invention.

Claims

1. A blockchain-based software distribution method for distributing a software package of a developer through a distributor, wherein a user of the software package obtains the same from the distributor through a package manager on the user's device, and a security provider accessible to the software package for security assessment, the method comprising:

receiving a request from the developer to obtain a witness contract for the software package from a Certificate Authority (CA), the request includes developer's account for declaring ownership over the software package;

obtaining evaluation from the CA a legitimacy and identity of the request through out-of-band procedures;

receiving the witness contract pertaining to the software package from the CA;

registering a hash value for the witness contract by making a call through a transaction which is mined on a blockchain;

receiving a security score of the software package based on a security assessment done by the security provider;

publishing the software package along with a verification proof, *proof*, wherein the proof comprises a block ID, proofs of the software package's registration, the developer's ownership, the security score and an expiry of the software package;

distributing the proof that is verified by obtaining a Markle tree proof to one or more databases in the blockchain.

2. The blockchain-based software distribution method according to claim 1, further comprising periodically receiving updated security score from the security provider of the software package.

3. The blockchain-based software distribution method according to claim 1, further comprising sending a copy of the software package to the security provided for security assessment to render the security score.

4. The blockchain-based software distribution method according to claim 1, wherein the security provider periodically provides security assessments to the software package and scores the results thereof, regardless of any changes to the software package by the developer.

5. The blockchain-based software distribution method according to claim 1, wherein all versions of the software package from the developer and the respective security scores are recorded on the witness contract.

6. A blockchain-based software distribution system for distributing a software package of a developer through a distributor, wherein a user of the software package obtains the same from the distributor through a package manager on the user’s device, and a security provider accessible to the software package for security assessment,

wherein the distributor operably performs any one of the claims 1-5.