A decentralized architecture for package manager based on the blockchain

Arnaud Stoz[1][0009−0008−0602−1159] and Thibault Debatty[1][0000−0003−2373−566X]

Royal Military School of Belgium

Abstract. Nowadays, package managers like *apt*, *pacman*, ... architecture relies on two main component to be function effectively: mirrors and package signatures. Those two component comes each with few drawbacks. The usage of mirrors put all the operating cost on mirror operators. This reduces the number of people able to participate in the system and thus is resiliency. The use of package signatures necessitates robust key management, ensuring that keys are securely maintained and never compromised. This paper presents a decentralized architecture designed to eliminate the need for package signatures while maintaining package integrity. Additionally, it aims to better distribute operating costs to enhance participation and system resiliency.

Keywords: Package manager · Decentralization · Blockchain · IPFS

1 Introduction

Today, the software supply chain has become a prime target for malicious threat actors, as demonstrated by recent attacks such as SolarWinds [14] and the XZ attacks [5]. From an attacker's perspective, compromising the software supply chain is highly valuable, as breaching just one stage of the supply chain can result in thousands of end-users being compromised. One of the main software supply chain stages is the distribution of the software to the end-user, which is typically based on a centralized architecture. This centralization and the usage of mirror introduces a vulnerability: if an attacker is able to impersonate a repository or set up a malicious mirror, he can launch a series of attacks (slow retrieval, replay old metadata, ...) targeting end-users 3.

Since the introduction of Bitcoin in 2008 [10], decentralized systems have gained popularity and robustness, establishing themselves as reliable frameworks for building trustable decentralized networks. The integrity of the system is maintained even if each participant has an incentive to cheat to maximize their own gains. Leveraging this technology and decentralized file systems, we can develop a new decentralized architecture that addresses the vulnerabilities inherent in centralized systems. This new architecture may improve latency performance, while also making it easier for end users to participate.

The contribution of this paper are the following:

– We review and compare the current architecture of two package managers: *apt* and *pacman*.

- We propose a new decentralized architecture relying on blockchain and decentralized files system.
- We compare the decentralized architecture with the existing centralized architecture.

The rest of this paper is organized as follows. Section 2 present terminology and definition of package manager, blockchain and decentralized storage used in the rest of this paper. Section 3 present the current security mechanism of package manager as well as identified attack vectors on package manager. Section 4 present the decentralized solution based on blockchain and decentralized storage. Section 5 compare the current centralized architecture with the proposed decentralized architecture. And finally, section 7 conclude this article and presents our future work.

2 Terminology and definition

2.1 Package Manager

End-users use package manager to install and update software (which is bundled together in a package) on their devices in an easy and convenient manner. In addition to downloading and installing software, a package manager keeps track and manages dependencies required for a given software to function properly. The way package manager manages the installation and update of software on the device is out of the scope of this article, but the interested reader can find more detail in [7].

2.2 Repository

Repository is used to centralized package location for the package managers. The repository generally functions as an FTP or HTTP(S) server, although it can also operate as a rsync server in some cases. Furthermore, it contains package metadata which allows the package manager to keep track of the current latest version of a software as well as the dependency tree of the software. Beside package metadata, a repository also contains:

- Root metadata
- Package metadata
- Software

The root metadata provides information on all packages available in the repository. It also includes details about the location of the package metadata file and the hash of this file to prevent tampering. The package metadata includes comprehensive information about the package, encompassing all dependencies required for the software to run smoothly. The software is the actual executable that the end-user intends to install and run. While this structure may vary slightly depending on the package manager used, the overall concept remains consistent.

2.3 Mirror

A mirror aids in distribution by providing redundancy, managing high load, and reducing latency through the duplication of repository content. Most of the time, the distribution repository will be operated by the distribution itself, while a mirror can be operated by anyone with enough disk space and bandwidth available to support the traffic. A mirror can be official and can be listed on the distribution website. Or it can be private and only people knowing the URL of the mirror will be able to use it. Packages are not supposed to be directly added to a mirror. It is only the reflection of the main repository.

2.4 Blockchain

The emergence of blockchain technology with the bitcoin brought a major evolution in the way a distributed system can be trusted. Thanks to the blockchain and more specifically the proof-of-work [9] consensus algorithm in the case of bitcoin, a distributed system where every participant would have interest in cheating to maximize their earnings, can be trusted, and all participants can find a consensus on a transaction modifying the current state of the system.

After Bitcoin, Ethereum [15], added a new functionality to the blockchain with smart contract. Smart contracts are pieces of code which can be executed by the Ethereum network. The new state reached after the execution of the smart contract made consensus and become the new state of the Ethereum network.

Interaction with a blockchain is never anonymous. All interactions are made by an account which is a private key signing on a behalf of a user. On a smart contract, specific accounts can be given specific rights on the contract. For instance, a call to a specific function in the contract can be restricted to only a small set of accounts.

2.5 Inter Planetary File System

IPFS [6] is a peer-to-peer protocol for storing and sharing data in a distributed file system. It relies on cryptographic hashes to store files. Unlike HTTP, IPFS is said to be content addressed instead of location based addressed. This mean instead of retrieving an asset based on a location (IP address, URL) we retrieve an asset based on the content of this file. This is achieved by query a unique hash linked to the file and its exact content. The immediate advantage is that the end-user is sure of the content he will receive. While on a location based system, if the server decides to replace the asset with another content but the same name, the end-user has no possibilities to detect this replacement or even get the previous version of the asset. However, the difficulty of retrieving IPFS stored content is that the end-user needs to remember a very difficult a human non-friendly hash.

3 Package manager security

One of the main concern of distribution is to make things very difficult for a malicious mirror to server a false or corrupted package. For all package managers, the primary security measure to mitigate this attack vector is the signing mechanism [11]. However, depending on which package manager is considered, the signature will not be on the same file. There are two different philosophies regarding which file should be signed:

- Package metadata signatures
- Root metadata signatures

Few package managers does not enforce any kind of security. It was the case of pacman before version 4.0.0 [3]. Before that version, pacman had no signature check capabilities. However, in version after 4.0.0, it's not enforced by default as pacman allows by default packages signed by keys in its local keyring as well as unsigned packages [4]. However, Arch Linux comes with a predefined pacman.conf file which configure pacman to install only verified signed package. For the rest of this article, pacman will be considered as enforcing and verifying package signature.

3.1 Package metadata signature

This category signs the package directly, which means only package signed by a trusted central authority can be downloaded on the device (e.g. pacman). Packages are directly signed by a trusted developer/package maintainer key and only those can be installed on a system.

3.2 Root metadata signatures

Instead of the package metadata, a package manager could sign the root metadata. The integrity rely on the chain formed by files hash to ensure the integrity of the data. The root metadata is signed by a trusted key. The first layer of security occurs at the root metadata directly. As this root metadata contains the list of all packages with their respective hash, any modification in the list package file will directly be noticed. As the root metadata is signed, it is impossible to modify its content. Therefore, the package list cannot be tampered with by an attacker, as altering it would require updating the hash in the root metadata file. The second layer of security comes from the list package manager containing the hash of the package itself. Modifying the package would require the attacker to modify the hash stored in the list package file and then modify the root metadata file. But again, without the key used to sign it's impossible. This is a cascade security. By signing only the top root metadata, all other files included in this root metadata file are protected from malicious modification.

3.3 Package repository attack vector

In this section, we summarize the findings of Justin Cappos et al. [8]. As the article is almost 20 years old, we will also quickly analyze if their findings are still relevant today or if package manager changed the way they worked. Based on this work, five major attacks possible on package manager can be listed:

- **Arbitrary package**: When an attacker can serve a complete arbitrary package to the user without the user being able to notice it. As now, almost all package managers implement signing mechanism, this attack become almost impossible unless the attacker controls the signing key.
- **Replay attack**: The attacker replays an older version of a correctly signed package.
- **Freeze attack**: Similar to a replay attack but instead of replaying an older correctly signed package, it just freezes the package to the current version.
- **Extraneous dependencies**: Rewrite the package metadata to install additional dependencies. Again, only feasible if the attacker has control over the signing key.
- **Endless data**: Return an endless stream of data to any query. This can crash the system.

Out of the five major attacks possible, two require the attacker to have knowledge of the signing key. While not impossible, we will consider this threat model as out of scope of this article. Indeed, we consider package manager protective and caution enough to correctly handle the signing key. We will use the above-mentioned attack as a baseline for the new architecture that we will propose. The new architecture should not be more exposed than what is explained in the Justin Campos et al. paper.

APT remark Even if HTTPS is supported, most mirror, included security mirror, are by default using HTTP. This opens the possibility of a sixth attack vector: DNS spoofing [13]. An attacker could redirect traffic to a malicious domain without the end-user noticing. If the mirror is already known by the end-user the potential damage is not important as the attacker could not modify anything without the signing key. However, if the attacker can trick the user to add his own key to the keyring, the attacker will have full control over software installed by the end user. It is particularly important when the end user adds a new repository as most of the time the public key used to verify the signature is stored on the server as well and added from the server itself to the keyring. This discussion is valid for every package manager but pacman seems to prioritize HTTPS when connecting to a mirror.

4 Decentralized architecture proposition

By combining the blockchain and the decentralized file system, we can propose a new architecture for package manager.

Thanks to the Blockchain and more especially smart contract, it becomes possible to move away from a signature mechanism while keeping the package integrity. The smart contract contains the logic that governs who is permitted to publish new packages. A quorum of administrators holds administrative rights on the smart contract to add or remove users authorized to push new packages. There are two advantages to this system:

1. It eliminates the need for the end-user to verify the package signature, as the package received from the smart contract can be trusted by default due to the blockchain's unfalsifiable properties.
2. Due to the distributed nature of the blockchain, it becomes easier for participants to join and leave the network at will. Although a certain number of fixed participants in the network is still necessary, this solution allows for a more "fluid" number of participants. Allowing this system to be more robust and reducing the latency.

However, this idea has a significant drawback. While blockchains are effective for storing transactions and achieving consensus, they are not designed to handle large volumes of data. Storing packages on the blockchain is impractical, and storing them in a smart contract is also inadvisable. This approach would significantly increase the size of the blockchain, thereby reducing the feasibility for end-users to participate in the network. To solve this issue, we will use another derivative technology called a decentralized file system and more specifically the Inter Planetary File System. Instead of storing the package on the blockchain, packages will be stored on IPFS and only their CID will be stored on the blockchain. The proposed architecture is illustrated on figure 1

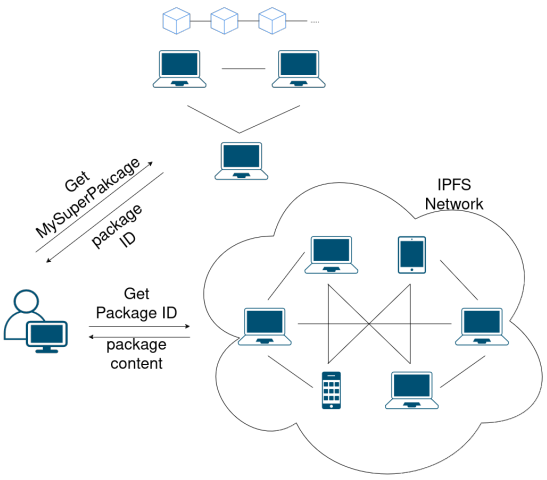

Fig. 1. Decentralized architecture

Once an account has been granted rights to push a new package to the system, it can create the package file as usual and then push it to the IPFS network. Once the package has been pushed a CID or package ID will be sent back from IPFS. This CID needs to be added in the smart contract mapping along with the name of the package for an easy retrieval for the end-user. The mapping will always be linked to a name which means the new version will overwrite the older version. This algorithm is described in algorithm 1.

Algorithm 1 Package upload algorithm

```
CID ← uploadPackageToIPFS
if CID ≠ nil then
    uploadCIDToBlockchain(CID, packageName)
end if
```

When an end-user wants to install a package, the query goes in the opposite way. It first queries the smart contract mapping to get the CID of the package. Once the CID is retrieved, it then queries the IPFS network to obtain the package. Before installing the package, CID must be checked. If the CID of the package received does not match with the CID received from the blockchain, the package received must be considered corrupted. The IPFS node which has served this package should be considered as malicious and blacklisted by the end-user. After blacklisting the node, the package can be queried again. This algorithm is illustrated in algorithm 2. The blockchain is used as a trusted de-

Algorithm 2 Package download algorithm

```
CID ← getCIDFromBlockchain(packageName)
receivedCID ← ""
if CID ≠ nil then
    while CID ≠ receivedCID do
        package ← GetPackageFromIPFS(CID)
        if CID ≠ CID(package) then
            BlacklistIPFSNode
        end if
    end while
end if
```

centralized system, primarily for its immutable properties, with smart contracts employed in conjunction to enhance its functionality. The smart contract acts as the guarantor. It stores the mapping between the name of the package and its associated CID on IPFS. It also serves as the identity management system by keeping track of who is authorized to publish packages. This approach eliminates the need for signing keys. A few accounts (the primary developer accounts) form an administrator quorum with the authority to approve new accounts for pack-

age submission to the system. The quorum also have the capability to remove accounts from the system.

5 Centralized vs decentralized

The current architecture of package manager is based on two main components:

- Replication of the repository through the help of mirrors.
- Private key to sign package.

We will review these two components and examine how they are impacted by the decentralized architecture.

5.1 Replication

Replication is the most significantly impacted component in the new architecture. The centralized architecture of the distribution can be visualized as a three-layer ring, as illustrated in Figure 2. At the center lies the distribution reposi-

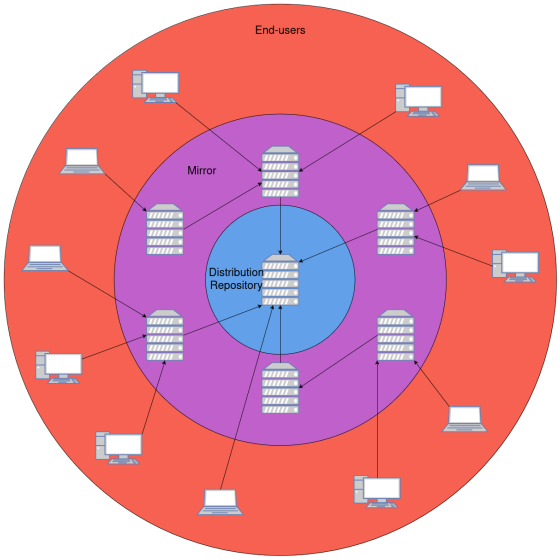

Fig. 2. three-layer ring infrastructure of package manager

tory. The second layer consists of all mirrors, while the third layer encompasses the end-users. Nevertheless, the mirror layer does not isolate the distribution repository from the end-users, they can also directly contact the distribution repository layer. There are two main drawback to this architecture compared to the decentralized solution proposed in this work:

1. Operating cost.
2. Mirror operator can behave maliciously.

Operating cost The mirror operator incurs costs in terms of disk space and bandwidth. The size of the Ubuntu package archive is about 2.6TB [2]. The mirror must be updated every six hours and must maintain high availability with sufficient bandwidth to handle synchronization with the distribution repository, as well as package downloads by end users. This restricts who is able to operate a mirror.

In the decentralized architecture, operating costs are lower because every user participates in the network. The load in terms of storage and bandwidth is distributed among more participants, making it easier for individuals to join the system and enhancing its overall resilience.

Malicious behavior The second drawback is the (relative) power it give to the mirror operator. As we have seen, it's unlikely a mirror operator can directly alter a package or serve a completely different package than the one asked by the end-user because the signing key is unknown to the mirror operator. However, that does not mean the mirror operator can not have a malicious behavior, as explained in section 3. If an end-user trusts a repository, they have limited means to easily detect whether the mirror is behaving genuinely or maliciously.

n the decentralized architecture, particularly with the use of blockchain, detecting malicious behavior becomes easier, as it only requires comparing what is served with what is stored on the blockchain.

5.2 Package signature

As explained in section 3, the security feature that protect end-users from modified packages is the use of signatures. This signature can be used to sign different file as discussed in section 3 (metadata, package). This implies that effective key management must be in place, as the private keys should never fall into the wrong hands. There is different approach for key managing depending on the philosophy of the package manager.

APT use only one key to sign a repository. The advantage is the easiness of use but obviously if a malicious actor gain access to this key it gains full control over the repository and can make any package looks legitimate. The process used by *apt* is illustrated on figure 3.

Pacman use a different approach. There is no single key to sign all package but use a web of trust model [1]. There are five different master keys with five different owners. The master keys are used to sign developer keys, which in turn sign packages. For a developer key to be considered valid, it must be signed by at least three master keys. In this approach, compromising a single master key does not grant any additional privileges to the attacker, as two more keys are still required to sign a package. The approach of *pacman* is illustrated on figure 4.

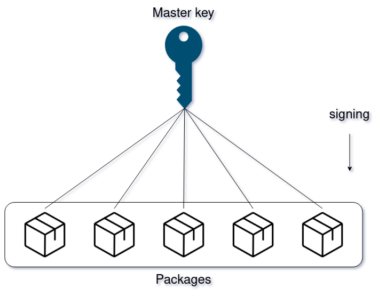

Fig. 3. The centralized approach

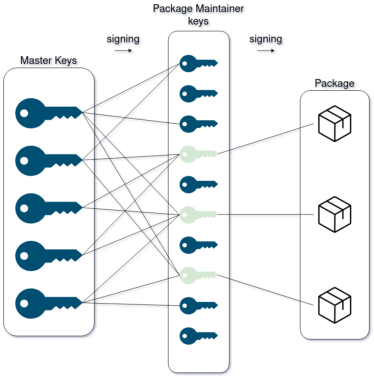

Fig. 4. The web of trust approach

The decentralized system tends to mimic the web of trust approach. The advantage of this approach is that compromising a single administrator in the quorum does not compromise the entire system, making it much more difficult for an attacker to gain full control. With the help of smart contract, it is easier to keep track, revoke and add who can push package to the system. All the logic is written in advance and nothing can alter it. This new logic is illustrated on Fig 5.

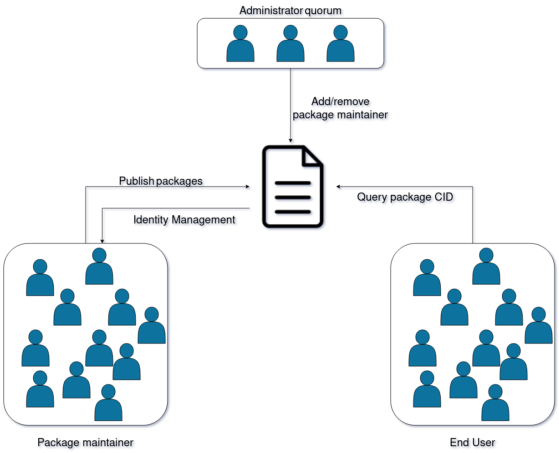

Fig. 5. Smart contract

6 Discussion

As explained in section 4, signatures are no longer required. The ability to push packages to the system replaces the need for signatures, as only a set of authorized users identified by their address can upload new packages. The integrity of the packages is ensured by the CID. There is no need to sign the package or the root metadata as it is taken care of by the system itself. It also makes it much easier for an individual to join the system. As shown in figure 1, the two networks, although they could be interleaved, are completely separated. This means the storage network (IPFS) is completely separated from the mapping network (the blockchain). A user can decide to participate in one or another depending on its resources. Both networks can even be much more dynamic than the current architecture of package manager. People could join and leave on a dynamic basis, like what is happening in BitTorrent [12]. Every user benefiting from the network should also contribute to it. The other advantage of using a decentralized file system is not all content must be synced. Each node contains only a part of the entire data available for download. While it's not impossible to have some "master“ node which always holds the entire content, this greatly reduces the amount of storage needed to participate in the network. All data replication and "master“ node are managed by the IPFS protocol itself to guarantee data reliability. This decentralization also offers an additional advantage: due to its distributed nature, it can enhance performance and reduce latency. The same idea applies for the mapping network (the blockchain network). Computers could join and leave in a dynamic way making the network more resilient.

As mentioned previously, it is essential to ensure that the new system meets at least the same security criteria as the current system. For the two attacks that required control over the signing key—namely, arbitrary package injection and extraneous dependency attacks—there is no change in the risk profile. Instead of controlling the signing key, an attacker would now need to control the account. One method to gain such control is through phishing attacks, which remains similar to current attack vector. For the endless data attack, the threat remains similar. If an attacker could return an endless stream of data either from the blockchain network or from the IPFS network, it could potentially crash the end-user system. Protection against this type of attack should be built into the client used to connect to the blockchain or IPFS. Finally, concerning the replay and freeze attack, those two become highly difficult to execute. As the blockchain will always hold the CID of the latest version of each package, the end-user could easily detect when a package is not the latest version available. As we can see, the new architecture meets the same security criteria and even mitigate two out of the five attacks vectors previously identified.

7 Conclusion

The current architecture of package manager has some drawbacks, from a security point of view but also on how it works and how people participate in the

system. We have proposed a distributed new system which gets rid of the signature mechanism by using blockchain and smart contract. Packages are stored on a decentralized file system IPFS in this paper. As we saw, this distributed system allows easier participation from end-users as well as mitigating two attack vectors found on the current architecture.

The work is ongoing for developing a proof-of-concept showcasing how this concept. This proof of concept should also answer some yet unquantified assumption. Main points this proof of concept should clarify are:

- The latency. As the system is now distributed and closer to the end-user it could improve the latency when downloading a package.
- The amount of storage used by IPFS node. As said before, the distributed nature of IPFS should allow reducing the storage needed by a peer in the network. However, this reduction should be quantified in regard to the space taken by a full mirror.

Another point to be considered is the consensus mechanism used in this system. Blockchain uses money penalties/reward to be sure every participant in the network behaves genuinely. However, in this case, we do not want to incentivize people purely with the help of money. We should think about another way to encourage people to behave genuinely on the network. While discussions are ongoing on this subject, for the proof of concept, the consensus mechanism will use either the proof of stake or the proof of authority.

References

1. The GNU Privacy Handbook, https://www.gnupg.org/gph/en/manual.html
2. Mirrors - Ubuntu Wiki, https://wiki.ubuntu.com/Mirrors
3. Pacman Home Page, https://pacman.archlinux.page/
4. pacman(8) — Arch manual pages, https://man.archlinux.org/man/pacman.8
5. XZ Utils backdoor, https://tukaani.org/xz-backdoor/
6. Benet, J.: IPFS - Content Addressed, Versioned, P2P File System
7. Bennett, S.P., Faulhammer, C., McCreesh, C., Müller, U.: Package Manager Specification
8. Cappos, J., Samuel, J., Baker, S., Hartman, J.H.: A look in the mirror: attacks on package managers. In: Proceedings of the 15th ACM conference on Computer and communications security. pp. 565–574. ACM, Alexandria Virginia USA (Oct 2008). https://doi.org/10.1145/1455770.1455841, https://dl.acm.org/doi/10.1145/1455770.1455841
9. Gervais, A., Glykantzis, V., Karame, G.O., Ritzdorf, H., Wüst, K.: On the Security and Performance of Proof of Work Blockchains
10. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System
11. Newman, Z.: Sigstore: Software Signing for Everybody. Los Angeles (2022)
12. Pouwelse, J.A., Garbacki, P., Epema, D.H.J., Sips, H.J.: THE BITTORRENT P2P FILE-SHARING SYSTEM: MEASUREMENTS AND ANALYSIS
13. Steinho, U., Wiesmaier, A., Araújo, R.: The State of the Art in DNS Spoong
14. Team, C.I.: SUNSPOT Malware: A Technical Analysis | CrowdStrike (Jan 2021), https://www.crowdstrike.com/blog/sunspot-malware-technical-analysis/
15. Wood, D.G.: ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER