Security analysis of Linux package manager

Analyzing and threat modeling of different package managers

Saad Ouassil Allak*, and Pranay Sarkar†
Fachbereich Informatik
Technische Universität Darmstadt
Hochschulstraße 10 , 64289 Darmstadt
* Email: saadouassil.allak@stud.tu-darmstadt.de
Matriculation Number: 2945765
† Email: pranay.sarkar@stud.tu-darmstadt.de
Marticulation Number: 2337328

***Abstract*—Package managers deals with the task of determining which packages are to be installed on a host and then downloading and installing all those packages along with all of their dependencies. There are different kinds of package managers available and all of them applies different security mechanism providing varying level of usability and resilience to different kind of attacks. Despite having existing security mechanism all of those package managers care vulnerable to man-in-the-middle (MIM) attack and malicious mirror. Security of package managers also depends on security practices of specific Linux distributions. When te distributions use third party mirrors as official mirrors, vulnerabilities can be easily exploited. It is also seen that when some security mechanisms control the location from where client gets the metadata and packages, actually decreases the system security. We explore the case of attacker having a compromised mirror and how it can compromise or crush thousands of clients, and we analyze the threat model using *STRIDE*.**

General Terms: Security
Keywords: Package manager, Package Management, Mirrors, Attack, Reply Attack, Man-in-the-middle (MIM), Threat Modelling, SDL, STRIDE

I. Introduction

PACKAGE managers are the most popular way for software distribution for all modern operating systems. *Package* is a software bundled into archives. Package managers provide centralized and privileged mechanism for software management in a system. As package managers needs superuser (i.e. $root$) access to install softwares, security of the installed packages are very important to secure the whole system.

This paper looks into some of the most popular package managers in Linux: APT [1], APT-RPM [2], YaST [4], YUM [5]. These package managers use one of the following four security models:

- No security,
- Cryptographic signatures embedded within the packages,
- Signatures on the detached package metadata,
- Signatures on the root metadata.

It can be seen that there is an ordering in the amount of security provided by different security models. Having no signatures allows the attacker to do most attacks, followed by having only package signatures, having signatures on detached package metadata and having signatures on root metadata. But there are some usability problems with some mechanisms. For example, signatures on root metadata do not provide a convenient way to verify *stand-alone package*, which might be obtained from a source other than the main repository. So in those cases users can install those packages even without using security checks. But those package managers which use signatures on detached package metadata or signatures on packages, can easily verify *stand-alone packages*.

It is recommended to combine two of these techniques together and create one layered approach for providing better security. (e.g. signatures on root metadata and signatures on packages or package metadata). This kind of layered approach was first added to Stork [6] package manager and now it is popular throughout the world.

Vulnerabilities are not always exploitable in real world. By looking into security structure of popular distributions, we found that it is necessary for an attacker to control an official mirror for package distribution system (e.g. Debian, Ubuntu, Fedora, OpenSUSE, CentOS). Otherwise attacker can not even launch attack on clients. Many distributions use mechanism for distributing requests to multiple mirrors or provide certain part of information from trusted source. But it can be seen from our analysis that it actually decrease the security of the system and thereby making the attacks easier.
Remaining part of the paper is organized as follows. In section 2, system arcitecture of package managers are described. Section 3 consist threat modeling, which includes types of attacks and different approaches to threat modeling. Section 4 describes our approach to threat modelling and application of *STRIDE*. Result of our threat modelling and comparison of effects are described in Section 5.

II. System Architecture of Package Managers

System architecture of package managers is shown in *Figure 1*.

From the system architecture diagram of package managers it can be seen that security offered by package managers varies depending on how signatures are used for data protection.

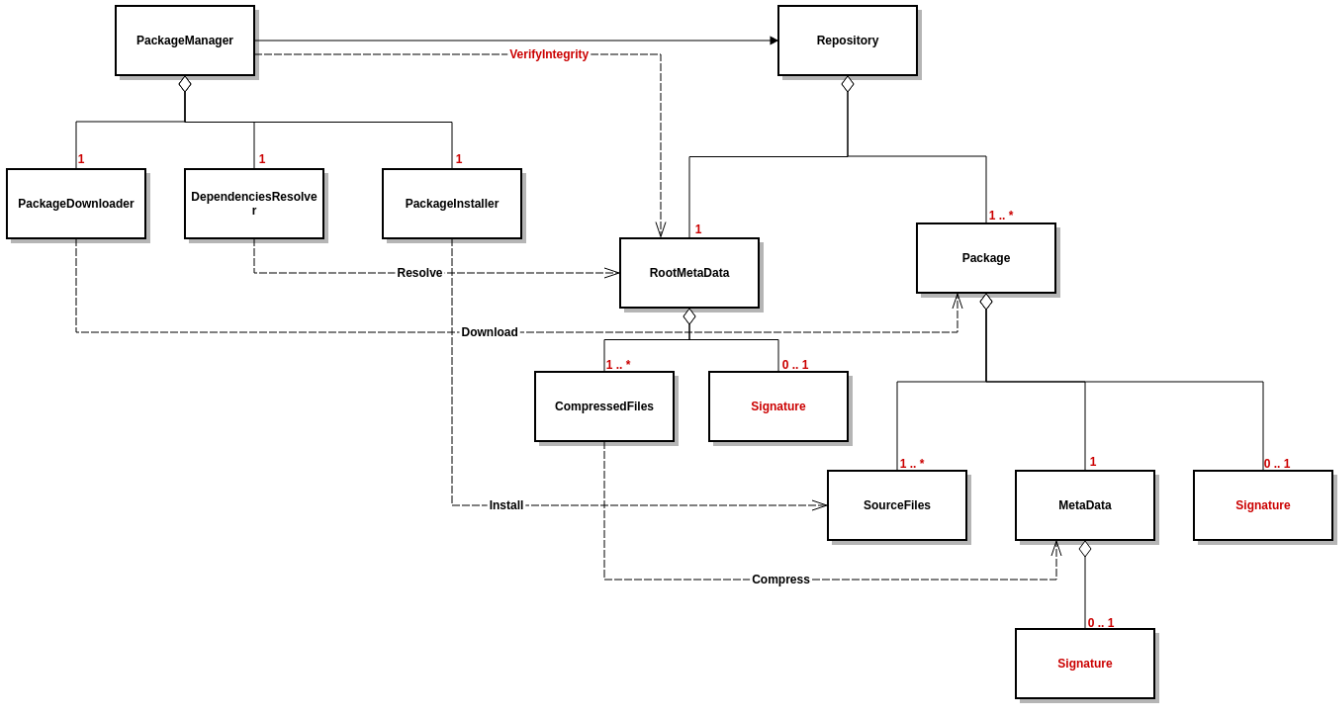

Fig. 1. System architecture for package manager

There are advantages and disadvantages of signatures in different data items. They are described in the following subsections.

Analyze model window as shown in *Figure 3* analyzes the given model and classifies the possible threat that can make the system vulnerable to possible attacks.

A. Package Managers with no security

There are some package manager that do not provide any security e.g. Slaktool. They do not sign any packages or package metadata or root metadata or root metadata file. So, any attacker that control a mirror can launch arbitrary package attack just by responding to client requests with malicious software.

B. Package Signatures

Package managers like YUM only relies on signatures embedded in package for providing security. Package metadata or root metadata are not protected. So the attacker can launch replay or freeze attack and the client will install vulnerable packages. There is another problem. YUM does not verify package metadata that they initially received for dependency resolution. As a result attacker can forge package metadata and launch extraneous dependencies attack. Endless data attack is also applicable to this kind of package managers.

C. Package Metadata Signatures

Some package managers like *Stork* can use signatures on package metadata. They can create a single file that contains time-stamp and securely hash the package metadata for all the packages that user trusts. Delegation of trust is also possible to single 'distribution' user. This kind of package managers are not vulnerable to extraneous dependency attack as the signatures protect the package metadata. But it is still vulnerable to endless data attacks. Attacker can also successfully launch freeze attack against this kind of package managers.

D. Root Metadata Signatures

Popular package managers like APT, YaST, APT-RPM use root metadata signatures. They can also support package signatures, but that feature is optional and not widely used in practice. Package metadata is stored in compress files and after securing all those file, hashes are stored in root metadata file. As root metadata is also protected with signature, package metadata is also protected from tampering. It prevents extraneous dependencies attacks. Another advantage of using root metadata is, it prevents a mirror from hosting more than one version of package from the main repository at the same time. Attacker have to choose time-period of main repository and it has to provide exactly same files when launching the replay attack. Unfortunately these package managers do not check timestamps. So, replay attack and endless data attacks are still possible.

III. Threat Modeling

Threat model involves attacker who can respond to legitimate requests made by a package manager. One example could

be man-in-the-middle *(MIM)*, where the attacker have tricked the client into contacting the wrong compromised server. It can also be a case where user have gained control over an official package distribution mirror. Threat model is described as follows:

- Attacker can send arbitary files to the user/client
- Attacker does know beforehand what the client is going to request
- Attacker does not have any trusted key to sign packages, package metadata or root metadata. As the mirrors does not get the private key to sign files as they only copy the already signed files from the main repository.
- Attacker have access to outdated packages, package metadata and root metadata. As there are many outdated repositories openly available to all.
- Attacker knows the vulnerabilities in some outdated packages and can exploit those vulnerabilities. Attacker can gain knowledge of this by going through the change-logs of updates or just by using some exploit toolkit.
- Attacker have no knowledge of vulnerability of latest version of package (i.e. zero-day vulnerabilities).
- If signatures are supported by package managers, it is used. But if there are any client or distribution who choose not to use signature supported by package managers, they are more vulnerable to attack.
- If supported and if current root metadata does not contain vulnerable versions of the package, expiration time for root metadata are used. As this root metadata is small file, it is feasible to sign it frequently.

A. Types of Attacks

Based on the above mentioned threat model, there can be many type of attacks which can be done on a client. All of the attacks can be used either to crash or control client's computer. Although the impact can vary for different types of attack, all of the attacks can be effective on some package managers:

- *Arbitrary Package:* Attacker provides a different package created by them in place of the real package to the user who wants to install it.
  Threats: Tampering.
- *Reply Attack:* Attacker replays older versions of request package which are correctly signed but contains security vulnerability. Attacker can then exploit the already existing vulnerability. It should be noted that package managers will never downgrade any existing package. So reply attack will not work while updating existing package. But it will work only when installing new package.
  Threats: Tampering, threats contained in previous release of package.
- *Freeze Attack:* Attacker freezes the information that the client sees to the current point of time, so that client can not see any more updates. It works in the similar way how freeze attack, where wrong metadata is provided to clients. The main goal of this attack is to compromise clients which already have installed vulnerable packages. This attack can also be used to prevent updates in addition of installing a out of date package.
  Threats: Tampering and threads contained previous release of package.
- *Endless Data:* Attacker returns endless stream of data in response to any download request from an already compromised mirror. This might result in package manager filling up the disk or memory on the client machine and thereby crushing it.
  Threats: Denial of Service (DoS) attack
- *Extraneous Dependencies:* Attacker overwrites package metadata so that addition packages have to be installed as a dependency along with the original package that the user want to install. If the package that is installed as a dependency have security vulnerability, it allows the attacker to compromise the system of the user. For example, if metadata of a package foo states that it depends on another package bar, it will cause the package bar to be installed even if it is not desired or needed. And, if this bar is vulnerable, the whole system can be compromised.
  Threats: Tampering

B. Approaches to Threat Modelling

Threat modelling can be approached in three different ways.

- *Attacker-Centric:* It mainly deals with the techniques of determining thee opponents (i.e. attackers). It also determines and categorizes the different ways by which attacks can act.
- *Software-Centric:* This kind of threat modeling tries to determine the possible ways by which the sofware and therefore the user is vulnerable to attacks. It also categorizes the types of attacks that can be performed on the specific software.
- *Asset-centric:* This kind of modeling determines what exactly to protect in the system. It can be different types of data. It also classifies the different levels of assets which can have different levels of effect on the system, if compromised to attackers.

IV. Approach & Tools used

We have mainly concentrated on threat modeling in software development. For that we have used *Microsoft SDL Threat Modeling*. It works in the following steps:

1) *Describe System:* It describes the individual components of the whole system, as well as the relationship and interaction between the components.
2) *Create Checklist:* After that a checklist if created. In this checklist all kind of possible attack scenarios are described for the overall system.
3) *Access impact and find countermeasures for each item:* The last step is to check the already created checklist, and measuring the possible impact of all possible attacks. Possible countermeasures are also be listed during executing this step.

We have described and analyzed all steps in threat model using *STRIDE*. We give them as an input to *STRIDE*and

*STRIDE*analyzes the model, ad generates the report automatically. We can also give the system model as n input to *STRIDE*.

As shown in *Figure 2* 'Main Administrator' can ada a package from main repository. It belongs to same trust domain. User can download package from Main repository or official mirror repository using package manager. Official main server is in the same trust boundary as of system administrator and thus it can be fully trusted.

A. Threat Model Report

*STRIDE*generates threat model report from the given input model. The report for package managers is as follows:

1) Multi-Process: This contains threats against setting up official mirror server. Main threat is tampering. It involves modifying integrity of packages. It can be mitigated by verifying package integrity. Continuous verification bs system administrator also helps.

2) Data Flows: This contains threats against downloading from mirror repository. There are two kinds of available threats: Tempering and Denial of Service. DoS can happen when mirror server can redirect download to another server cause congestion to the other server. This can be mitigated by verifying the size of packages in manager level.

Tampering can happen when there is a corrupt official mirror. It depends on the type f verification that package manager does. Verifying the integrity of package manager helps a lot.

3) Data Stores: It contains threats against mirror repository. Possible problems can arise by tampering, Repudiation, Information disclosure and denial of service attacks.

Data tampering can happen when any compromised mirror repository have some corrupted or maliciously changed data. It can be solved ba periodical checking.

Repudiation happens if there is some old or compromised mirror who sends vulnerable packages to the system. Solved by periodical synchronization between main repository and mirror repositories. Denial of service attack happens when there is endless data attack.

V. Comparison

We have analyzed different linus distributions and the security mechanisms they provide. Their effectiveness against attacks are also variable.

- *SUSE Enterprise Linux:* Being a commercial distribution it does not support any mirrors hosted by outside parties. So it is not vulnerable to all the attacks described in the paper.
- *OpenSUSE:* It uses download redirector, which provides protection for malicious mirrors. But is any man-in-the-middle (MIM) attacker responds to client requests, then it can perform replay or freeze attack. If there is any denial of service (DoS) attack abd the download redirector fails or becomes unreachable, users can not get any update. But it also removes the risk of replay attack. Users are always vulnerable to endless data attack from compromised mirror.

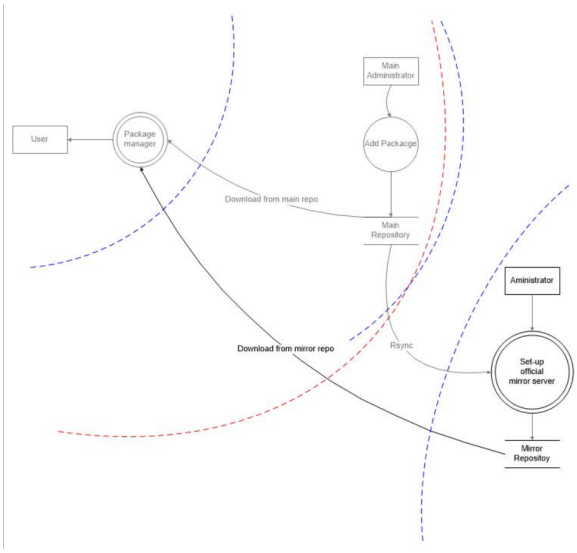

Fig. 2. System threat model diagram

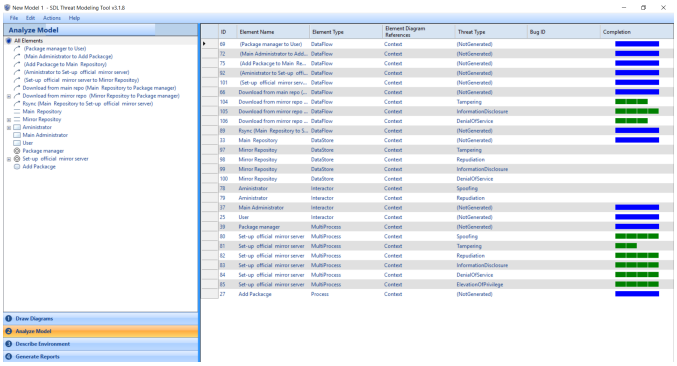

Fig. 3. Result of analyzing model in *STRIDE*

- *Ubuntu:* It have all the problems of OpenSUSE and one additional problem. If the security repository fails, the become vulnerable to replay or freeze attack from the compromised mirrors.
- *Debian:* Being a community driven distribution like Ubuntu and OpenSUSE, it have all the problems faced by Ubuntu.
- *Red Hat Enterprise Linux:* It does not face any security threat from malicious mirrors since it does not use any mirrors hosted by outside parties, just like SUSE Enterprise Linux. It is vulnerable to man-in-the-middle attack because of the flaw in their HTTPS implementation.
- *Fedora:* Even if attacker got access to a mirror, download redirector makes it more difficult to launch attacks for the malicious mirrors. The reason is, the malicious mirrors needs to provide snapshots of packages to the download redirector. But endless data attack is still possible. Another problem is attacker can target attacks to specific IP range.

VI. Related Works

There are some other package managers apart from linux package managers. We have also looked into them to see how

they works.

- *pip: The python package manger:* It checks only the dependencies between the package that is about to be installed and already installed packages. Connection is not encrypted between client and the server. Advantage is there is no mirror. So, the attacker had to person man-in-the-middle (MIM) attack to send vulnerable packages to the client.
- *npm: The NodeJS package manager:* It works almost same as pip, the package manager for python. There are no mirrors and there iy only official repository to download from. All the dependencies are checked before accepting any package into npm library. If the server goes down for some reason such as denial of service attack, users just can not install anything.
- *Android app packaging at Google Play store:* Android app uses specific permission manager for all apps. In $persistance.xml$ the ap developer have to write all dependencies and all the possible permissions that the app can take during it's execution. Google play store provides centralized downloading for apps. The connection between play store services and the user is encrypted. So, freeze or reply attack will not work. Man-in-the-middle (MIM) attack may succeed, but eve if the attacker tricks the user into downloading some wrong app, while installing the app the Android permission manager will show all the necessary permission. If the user see there are some difference from what it should be or if there is any suspicious permission, she might not install it. Which solves the problem.

VII. Conclusion

Our work identifies different security issues in some of the most popular Linux package managers that is sued today. By using threat modeling we show all possible points where attack can happen. *STRIDE*also helps us to understand in details all the possible attacks without even executing the system in real-time. And it can also be seen that some security mechanisms used by distributions helps us to prevent attacks and there are some of them which increases the chance of attack by decreasing the security of the whole system.

Acknowledgment

The authors would like to thank Lotfi ben Othmane (*lotfi.ben.othmane@sit.fraunhofer.de*) for providing us the opportunity to work on the mini project in Secure Software Development (SecDev).

References

[1] Debian APT tool ported to Red Hat Linux, https://wiki.debian.org/apt-get/
[2] APT-RPM. http://apt-rpm.org/
[3] Arch Linux (Don't Panic) Installation Guide, http://www.archlinux.org/static/docs/arch-install-guide.txt
[4] YaST - openSuSE. http://en.opensuse.org/YaST
[5] Yum: Yellow Dog Updater Modified. http://linux.duke.edu/projects/yum/
[6] Stork. http://www.cs.arizona.edu/stork
[7] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.