A look in the mirror: Attacks on package managers

Conference Paper · January 2008

DOI: 10.1145/1455770.1455841 · Source: DBLP

4 authors, including:

John H. Hartman
The University of Arizona
61 PUBLICATIONS **1,992** CITATIONS

SEE PROFILE

A Look In the Mirror: Attacks on Package Managers

Justin Cappos Justin Samuel Scott Baker John H. Hartman
Department of Computer Science, University of Arizona
Tucson, AZ 85721, U.S.A.
{justin, jsamuel, bakers, jhh}@cs.arizona.edu

ABSTRACT

This work studies the security of ten popular package managers. These package managers use different security mechanisms that provide varying levels of usability and resilience to attack. We find that, despite their existing security mechanisms, all of these package managers have vulnerabilities that can be exploited by a man-in-the-middle or a malicious mirror. While all current package managers suffer from vulnerabilities, their security is also positively or negatively impacted by the distribution's security practices. Weaknesses in package managers are more easily exploited when distributions use third-party mirrors as official mirrors. We were successful in using false credentials to obtain an official mirror on all five of the distributions we attempted. We also found that some security mechanisms that control where a client obtains metadata and packages from may actually decrease security. We analyze current package managers to show that by exploiting vulnerabilities, an attacker with a mirror can compromise or crash hundreds to thousands of clients weekly. The problems we disclose are now being corrected by many different package manager maintainers.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Invasive software; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.4.1 [**Social Issues**]: Abuse and Crime Involving Computers

General Terms

Security

Keywords

Package Management, Mirrors, Replay Attack

1. INTRODUCTION

Package managers are a popular way to distribute software (bundled into archives called packages) for modern operating systems [1, 2, 3, 21, 22, 25, 26, 28, 31, 32]. Package managers provide a privileged, central mechanism for the management of software on a computer system. As packages are installed by the superuser (root), their security is essential to the overall security of the computer.

This paper evaluates the security of the eight most popular [9, 19] package managers in use on Linux: APT [1], APT-RPM [2], Pacman [3], Portage [21], Slaktool [25], urpmi [28], YaST [31], and YUM [32]. Also examined is the popular package manager for BSD systems called ports [22] and a popular package manager in the research community called Stork [26]. These package managers use one of four different security models: no security, cryptographic signatures embedded within packages, signatures on detached package metadata, or signatures on the root metadata (a file that contains the secure hashes of the package metadata).

This work demonstrates that there is an ordering to the amount of security provided by the different package manager security models. This order is preserved even as other security weaknesses in the package managers are corrected. Having no signatures allows the most egregious attacks, followed by having only package signatures, having signatures on detached package metadata, and finally having signatures on the root metadata, which provides the most security. However, there are usability concerns with different signature mechanisms, most notably the ability to verify a stand-alone package (a package obtained from a source other than the main repository). Signatures on root metadata do not provide a convenient way to verify stand-alone packages and so the user is likely to install such packages without using security checks. In contrast, package managers that use signatures on detached package metadata or signatures on packages can verify stand-alone packages.

Because of the usability strengths and weaknesses of different techniques for providing security, this work recommends a layered approach created by combining two techniques: signatures on the root metadata and either signatures on packages or package metadata. This technique provides the security strengths and the usability strengths of both types of signatures with an overhead of between 2-5%. The layered approach has been added to the Stork package manager and is now in use by thousands of clients around the world.

While we find that vulnerabilities in package managers do exist, vulnerabilities are not always exploitable in the real world. We examine this by looking at the security of popular distributions. We find that it is trivial for an attacker to control an official package mirror for a popular

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

distribution (such as Ubuntu, Debian, Fedora, CentOS, and openSUSE) and therefore to be in the position to launch attacks on clients. To mitigate this threat, many distributions use mechanisms to distribute requests to multiple mirrors or provide certain information from a trusted source. We show that some of these mechanisms actually decrease the security of users by making it easier to target attacks.

2. BACKGROUND

2.1 Package Formats

Packages consist of an archive containing files and, in most cases, additional *embedded package metadata*. For a given package, the embedded package metadata contains information about any other packages that must be installed in order for it to operate (the *dependencies*), functionality the package possesses (what the package *provides*), and various other information about the package itself. The most popular package format, RPM [23], has space for one signature. Other popular package formats have no standard field for signatures, although in some cases extensions exist to support signatures [8, 11].

2.2 Package Managers

Clients use a package manager to install packages on their system. A package manager gathers information about packages available on package repositories. Almost all package managers automatically download requested packages as well as any additional packages that are needed to correctly install the software. This process is called *dependency resolution*. For example, a requested package `foo` may depend on `libc` and `bar`. If `libc` is already installed, then `libc` is a dependency that has been resolved (no package must be installed to satisfy the dependency). If there is no installed package that provides `bar`, then `bar` is an *unresolved dependency* and a package that *provides* `bar` must be installed before `foo` may be installed. The package manager may be able to locate a package that provides `bar` on a repository.

The packages that are chosen to fulfill dependencies may have unresolved dependencies of their own. Packages are continually added to the list of packages to be installed until either the package manager cannot resolve a dependency (and produces an error) or all dependencies are resolved.

2.3 Repository

A package repository is usually an HTTP or FTP server from which clients can obtain packages and *package metadata*. The package metadata is usually just a copy of the embedded package metadata for all packages on the repository. Package managers download the package metadata from a repository so that they know which packages are available from that repository. This also provides the package manager with dependency information needed to perform dependency resolution. To facilitate convenient downloading of package metadata, most repositories store all of the package metadata in a small number of compressed files.

In addition to the package metadata, almost all repositories have a *root metadata* file. The name and location of the root metadata file varies for different repository formats, but the contents are similar. The root metadata provides the location and secure hashes of the files that contain the package metadata.

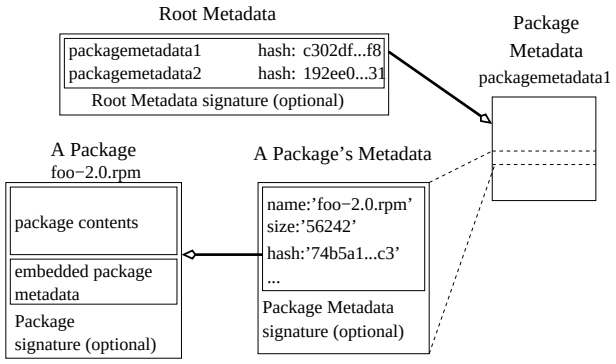

Figure 1: Repository Layout. The root metadata, package metadata, and packages may all optionally have signatures depending on the support of the package manager. Arrows point from a secure hash to the file it references.

Figure 1 shows the layout of a typical repository. A package manager downloads the root metadata and uses that to locate the files containing the package metadata. The package manager then downloads the package metadata. The package metadata is used to determine package availability as well as for dependency resolution. Packages are then downloaded and installed. The root metadata, package metadata, and packages may be signed depending on the security model of the package manager.

2.4 Mirror

It is common for a distribution to have more than one server from which users can download packages and package metadata. There is usually a *main repository* for a distribution whose contents are copied by many separate *mirrors*. A mirror typically contains exactly the same content as the main repository and is updated via `rsync` or a similar tool. A mirror differs from a main repository in that a mirror is not intended to have packages directly added to it or removed from it by its administrators. Packages are added or removed only on the main repository and the mirrors later obtain the changes when copying the main repository.

A mirror can be *public* (available for anyone to use) or *private* (restricted to a specific organization). A mirror may also be endorsed by a distribution for public use, typically when that the distribution is in contact with the mirror maintainers. This type of mirror is called an *official* mirror (the terminology used outside of this document varies by the distribution). Official mirrors are by definition public because the distribution is endorsing their use to the public.

It should be noted that some distributions do not use official mirrors hosted by outside organizations. One example is tiny distributions that can support all of their clients by a small number of repositories that the distribution directly controls. Another example is a distribution that requires users to pay. These costs often are used to support a set of internally maintained mirrors for the distribution. Alternatively, a distribution may allow or require each organization using the distribution to set up their own private mirrors for the organization's own use.

However, official mirrors hosted by outside organizations are the predominant mechanism for software distribution with all but two popular distributions [9, 19] relying on offi-

cial mirrors. Official mirrors are essential for most distributions to reduce cost and management overhead.

3. THREAT MODEL

The threat model used in this paper involves an attacker that can respond to requests made by a package manager. This can be a man-in-the-middle, an attacker that has tricked a client into contacting the wrong server (e.g. through DNS cache poisoning), or an attacker who has gained control of an official mirror for a distribution. The threat model is as follows:

- The attacker can serve the client arbitrary files.
- The attacker does not know what package the client will request a priori.
- The attacker *does not* have a key trusted to sign packages, package metadata, or the root metadata. — Note that mirrors do not usually possess the private key used to sign files, they only copy previously signed files from the main repository.
- The attacker has access to outdated packages, outdated package metadata, and outdated root metadata files. — There are many outdated mirrors on the Internet where an attacker can obtain these files.
- The attacker is aware of vulnerabilities in some outdated packages and is able to exploit those vulnerabilities. — This is possible by looking at change logs and updates to software source files or downloading an exploit toolkit [18].
- The attacker does not know of a vulnerability in the latest version of any package. — Zero-day vulnerabilities are unlikely to be known by many attackers.
- If a package manager supports signatures, signatures are used. — If a client or distribution chooses not to use signatures supported by their package manager, they are as vulnerable as if they used a package manager that does not support signatures.
- Expiration times in the root metadata are used, if supported, and current (un-expired) root metadata does not contain any vulnerable versions of packages — The root metadata is a single, small file so it is feasible for the main repository to sign it relatively frequently with short expiration times.

3.1 Attacks

Given this threat model, there are several attacks that may be used on a client. The impact of these attacks varies, but all allow the attacker to either crash or control the client's computer (possibly via exploiting a package with a known vulnerability). Each of the following attacks is effective on at least some of the package managers we studied:

- *Arbitrary Package* The attacker provides a package they created in place of a package the user wants to install.
- *Replay Attack* An attacker replays older versions of correctly signed packages or metadata, causing clients to install an old package with security vulnerabilities the attacker can exploit. The attacker can then compromise the client by exploiting the vulnerable package. Note that most package managers will not downgrade an existing package, so a replay attack only works when the package manager installs a new package, not when it updates an existing package.
- *Freeze Attack* Similar to a replay attack, a freeze attack works by providing metadata that is not current. However, in a freeze attack, the attacker freezes the information a client sees at the current point in time to prevent the client from seeing updates, rather than providing the client older versions than the client has already seen. As with replay attacks, the attacker's goal is ultimately to compromise a client who has vulnerable versions of packages installed. A freeze attack may be used to prevent updates in addition to having an installed package be out of date.
- *Extraneous Dependencies* The attacker rewrites the package metadata to have additional packages installed alongside a package the user intends to install. For example, the attacker provides metadata that incorrectly states that package `foo` depends on `bar`. This will cause `bar` to be installed when it is not desired or needed. If `bar` has a security vulnerability, this allows an attacker to compromise the client's system.
- *Endless Data* This attack is performed by returning an endless stream of data in response to any download request. This may cause the package manager to fill up the disk or memory on the client and crash the client's system.

4. SECURITY OF PACKAGE MANAGERS

The security of a package manager varies depending on how signatures are used to protect data. This section explores the security strengths and weakness of signatures on different data along with implementation pitfalls observed in package managers (and how to fix them). This section then classifies the security of different signatures into a list ordered by increasing security.

The discussion groups package managers with similar security characteristics together. The first group of package managers do not use signatures (Section 4.1). The second group of package managers use signatures on packages but do not use signatures on package metadata or root metadata (Section 4.2). The third group of package managers use signatures on package metadata but not on the root metadata (Section 4.3). The final group of package managers use signatures on the root metadata (Section 4.4).

4.1 Package Managers Without Security

There are three popular package managers that do not provide security: Pacman, ports[1], and Slaktool. These package managers do not sign packages, package metadata, or the root metadata file. As a result, any attacker that controls a mirror can trivially launch an arbitrary package attack by responding to client requests with malicious software.

[1]A version of ports used by NetBSD did support package signatures at one time [7], but this has been obsoleted and is not maintained or used.

4.2 Package Signatures

YUM and urpmi rely solely on signatures embedded in packages to provide security. There is no protection of package metadata or the root metadata. As a result, an attacker can launch replay or freeze attacks and have clients install vulnerable packages. An attacker can choose to include any vulnerable versions of signed packages they wish on the repository.

However, neither YUM nor urpmi verify that the package metadata they initially received for dependency resolution matches the embedded package metadata of subsequently downloaded packages. This allows an attacker to forge package metadata to launch an extraneous dependencies attack. This means that if there exists a package that the attacker knows how to exploit and which the user does not have installed, the attacker can cause it to be installed whenever any other package is installed by the user. The result is that an attacker can compromise essentially every client that installs or updates a package.

Both YUM and urpmi are also vulnerable to endless data attacks. For example, when YUM is given a `repomd.xml` file of unlimited size, it exits without printing an error after the filesystem is full — leaving the huge file on disk. Since no information is logged or printed about the error, this makes discovering the problem complicated (especially if YUM runs via auto-update).

Fixing the package managers Even without changing the signature methods, it is possible to modify these package managers to prevent the extraneous dependencies attack. Doing so requires verifying that downloaded package metadata is the same as the embedded metadata in any downloaded package. Furthermore, endless data attacks can be prevented by capping the size of downloaded data. However, these changes are not adequate to protect against replay or freeze attacks. Given the large number of packages that need to be re-signed to prevent replay or freeze attacks, we believe that adding root metadata signing is the most practical way to address this issue.

4.3 Package Metadata Signatures

The Portage and Stork package managers use signatures on package metadata; however, they do so in different ways. Each package in Portage has a separate, signed package metadata file for each version of the package. The package metadata contains the secure hash of the package (possibly along with hashes of related files such as patches). In contrast, Stork users create a single file that contains a timestamp and the secure hash of the package metadata for all of the packages that the user trusts. Users can also delegate trust to other users and all users typically delegate trust to a single "distribution" user. The analysis of Stork therefore focuses on the security of the packages trusted by the distribution user because the security of the distribution user affects all clients.

These package managers are not vulnerable to extraneous dependencies attacks because the signatures protect the package metadata. However, resistance to metadata tampering does not imply that the package manager is resistant to all attacks. Both package managers are vulnerable to endless data attacks.

In both Portage and Stork, an attacker can launch freeze attacks. In Portage, since each package has a different file for metadata signatures, an attacker can choose to have any combination of packages (such as those that include only older versions with known vulnerabilities) available on the mirror. Portage is vulnerable to replay attacks in the same way as freeze attacks.

In Stork, all of the package metadata hashes are in the same signed file. This prevents an attacker from choosing package metadata that existed on the repository at different times. Stork checks that timestamps on files are increasing to prevent replay attacks but does not prevent freeze attacks.

Fixing the package managers It is possible to add replay attack protection to Portage through timestamp checking and to add protection against endless data attacks to both Portage and Stork. However, both package managers will still be vulnerable to freeze attacks.

4.4 Root Metadata Signatures

The package managers APT, APT-RPM, and YaST use signatures on the root metadata. All three of these package managers optionally support package signatures as well, but this functionality is not widely used in practice.

Package metadata is stored in compressed files and the secure hashes of those files are stored in the root metadata file. As the root metadata is protected by a signature, the package metadata is protected from tampering which prevents extraneous dependencies attacks. In addition, the signature on the root metadata prevents a mirror from hosting versions of packages that were on the main repository during different time periods. The attacker must choose a time period of the main repository to copy and provide exactly those files when launching a replay attack. Unfortunately, none of these package managers check the order of timestamps to prevent replay attacks or have any mechanisms to prevent freeze attacks or endless data attacks.

Fixing the package managers It is easy for these package managers to protect against endless data attacks as well as to prevent replay attacks. Replay attacks can be prevented by adding a timestamp to the root metadata and checking that any newly downloaded root metadata is not older than the version the client last obtained. In fact, all three package managers have a timestamp available and merely need to add this check.

To mitigate the effectiveness of freeze attacks, package managers could add an expiration time to the root metadata. Clients would refuse to use a root metadata file if the current time is greater than the expiration time. Since the root metadata is a single, small file, it is feasible to re-sign this file often and require every mirror to be frequently updated (most distributions already require their public mirrors to update at least once a day).

4.5 Classification

The security mechanisms and vulnerabilities of the package managers are summarized in Figure 2. All of the package managers studied are vulnerable to endless data attacks as well as freeze attacks. Different package managers have varying resistance to other attacks such as replay attacks. Depending on the package manager's security mechanisms, the result can be any of the following, where those listed first also imply those listed after: *arbitrary* packages created by the attacker are installed, any vulnerable package can be installed *alongside* non-vulnerable packages a client installs using an extraneous dependencies attack, *mismatched* outdated packages are installed (in that they existed on the

Name	Signature	Package Installation	Metadata Abuse
Pacman	nothing	arbitrary	arbitrary
ports	nothing	arbitrary	arbitrary
Slaktool	nothing	arbitrary	arbitrary
YUM	(1)	alongside	arbitrary
urpmi	(1)	alongside	arbitrary
Portage	(2)*	mismatch	replay / freeze
Stork	(1)*, (2)	snapshot	freeze
APT	(1)*, (3)	snapshot	replay / freeze
YaST	(1)*, (3)	snapshot	replay / freeze
APT-RPM	(1)*, (3)*	snapshot	replay / freeze

Figure 2: Package managers, their protection mechanisms and vulnerabilities. The protection mechanisms are numbered (1) packages, (2) package metadata, (3) root metadata. '*' indicates that support exists but is not in common use.

Signatures Protecting	Best Case		Common Case	
	Package	Metadata Abuse	Package	Metadata Abuse
No Security	arbitrary	arbitrary	arbitrary	arbitrary
Package	mismatch	replay / freeze	alongside	arbitrary
Package Metadata	mismatch / snapshot	freeze	mismatch	replay / freeze
Root Metadata	current	none	snapshot	replay / freeze

Figure 3: Classification of package manager protection schemes. This demonstrates both the security that is possible to achieve using a scheme as well as what is commonly provided by existing implementations.

main repository at different times), or installed packages will come from a collection of outdated packages that all existed at the same time on the main repository (i.e. in the same *snapshot*).

Based on the observation and analysis of the security in existing package managers, it is possible to similarly classify the security mechanisms. As Figure 3 shows, one can obtain an ordering of the security of the mechanisms. Clearly, having no signatures allows the most attacks and is the most vulnerable. Signatures on packages provide a definite improvement over no signatures, but gives the attacker the ability to manipulate metadata arbitrarily and provides attackers the ability to populate a mirror with packages of mismatched versions, or, if package metadata isn't verified using the signed packages, the ability to cause vulnerable packages to be installed alongside any non-vulnerable packages. Signatures on package metadata prevent the attacker from doing more than replaying or freezing the package metadata, but if the signatures are in separate files, the attacker can still mismatch versions of packages. By preventing replay and freeze attacks in package managers that sign the root metadata, a package manager will only install current packages and is immune to metadata tampering.

5. ADDITIONAL USABILITY NEEDS

This section focuses on additional usability requirements users have for package management. Most importantly, the use case where a user has an uninstalled package on their computer they need to verify.

The standard use case of the package managers and their security mechanisms is where a user needs to securely install software from a repository or mirror. However, it is not uncommon for a user to have a stand-alone package that was created by a party they trust and that they need to verify is free from tampering. Stand-alone packages are packages that are not obtained through the package manager's normal channels at install time. Stand-alone packages may have been obtained manually from unofficial sources or may even be packages a user has created. Another source provides an extended discussion [29] describing why stand-alone package verification is an important and desirable feature.

The signing of only root metadata does not allow any practical way to verify stand-alone packages. Package managers that use signed root metadata could be modified to keep copies of all metadata obtained from the repository for future verification of stand-alone packages, but this only helps for packages a user manually downloads from the same repository that they access through their package manager. This also fails to satisfy one of the primary reasons given for being able to verify stand-alone package signatures: verifying signatures for files when they are only available for manual download and installation, not through a repository.

Package managers that sign package metadata tend to be more able to meet the needs of stand-alone package verification than the package managers who only sign root metadata. However, the way in which package metadata is stored has a major impact on usability in this case. Similarly with package managers that sign only root metadata, package managers would need to store old package metadata and this would only be of use for verifying stand-alone packages that came from a repository the user normally uses. In other cases, the user would need to be sure to always keep the signed package metadata with the package for verification purposes. This is far from an ideal option.

Signatures embedded in packages are thus the most practical option and provide the greatest ease of use when stand-alone packages must be verified. All that a user needs in order to verify a package is the package itself. A drawback with having signatures in the package is that signatures are constrained by the limits of the package format so multiple signatures may not be supported.

Using signatures embedded in packages for stand-alone package verification does have complications, though. Notably, users must have the requisite public keys available in order to verify package signatures. They must also ensure on their own that packages they are installing are not outdated or have vulnerabilities. However, there are many scenarios where a user can use embedded package signatures in a way that increases security in their specific situation.

6. DEPLOYMENT EXPERIENCE

To gain more experience with what security mechanisms work well in practice, we modified the package manager Stork and added root metadata signing. We added an expiration time to the root metadata to prevent freeze attacks. Since Stork already supported both package signatures and package metadata signatures, this allowed us to experiment with all types of signatures in a single package manager.

The changes to Stork for root metadata signing were tested and beta-deployed and finally incorporated into the production release. Interestingly, Stork differed from all other secure package managers in that there was no key already trusted by clients to validate communication from the repository. As the only signed files in Stork were the package

metadata files signed by individual users, there had never been a need for the repository to have its own key that the clients trusted. This required distributing a repository key to clients in order for them to make use of the new root metadata signatures. The key was included with the initial release of the updated version of Stork. This initial key distribution was secure because the majority of users, through their trusted packages files, delegate trust to the Stork team to provide them updated Stork packages. Stork's design meant that users would not be using this key for trusting packages, but rather only verifying metadata files downloaded from the repository.

The resulting changes were transparent to the users both in terms of performance and usability. The overhead of using package metadata signing along with root metadata signing was measured and found to be negligible (between 2-5%). Ultimately, there were few comments about the addition of root metadata signatures since the existing security mechanism (package metadata signatures) was retained without modification. Though transparent to the users, they gained increased security through the addition of root metadata signatures.

We examined packages on the Stork repository to find that user-uploaded packages did not include package signatures, indicating that researchers were not using the optional package signature feature of Stork. While far from conclusive, this implies that package metadata signatures and package signatures are redundant. We reason that when multiple signature mechanisms are provided, either package metadata signatures or package signatures are sufficient for usability purposes, but it is unnecessary to support both.

To conclude, since no one scheme works well from both a security and a usability standpoint, we propose that package managers should use multiple security mechanisms. It is clear that root metadata signatures should be included because of their security benefits. It also seems advantageous to have either package metadata signing or package signatures for usability. By combining root metadata signatures with either signed package metadata or signed packages, a package manager can obtain a high degree of security and excellent usability without significant performance impact.

7. PACKAGE MANAGERS IN PRACTICE

This section examines additional functionality that is provided by some of the most popular distributions (Ubuntu, Debian, Red Hat Enterprise Linux, Fedora, CentOS, openSUSE, and SUSE Enterprise Linux) that impacts the security of package managers in practice. It should be noted that we did a cursory examination of other popular distributions (Gentoo, Mandriva, LinuxMint, Sabayon Linux, Slackware, KNOPPIX, Arch Linux, and MEPIS Linux) and did not find any additional security practices that significantly altered the security over that provided by their package manager. We did not survey any smaller distributions, but we strongly suspect that the majority of them are at least as vulnerable as their package manager is by default.

In this section, we first examine the feasibility of an attacker obtaining an official mirror for a distribution. We examine this by setting up mirrors for five popular distributions that have official mirrors run by outside parties (Ubuntu, Debian, Fedora, CentOS, and openSUSE). We then examine the security pitfalls in using a security repository for package updates and a set of untrusted mirrors for the core distribution. Next, we examine how the mechanisms by which requests are distributed to different mirrors impacts security. We then examine the use of HTTPS to try and determine its effect and applicability. The section concludes with a look at the discussed distributions and a comparison of their security characteristics.

7.1 Obtaining a Mirror

To evaluate the feasibility of controlling mirrors of popular distributions, we set up public mirrors for the CentOS, Debian, Fedora, openSUSE, and Ubuntu distributions. A fictitious company (Lockdown Hosting) with its own domain, website, and fictitious administrator (Jeremy Martin) were used as the organization maintaining the mirrors. A server with a monthly bandwidth quota of 1500 GB was leased for $200 per month through The Planet (`www.theplanet.com`).

Setting up a public mirror for each distribution involved acquiring the packages and metadata from an existing mirror and then notifying the distribution maintainers that the mirror was online and available for public usage. The distributions varied in terms of the degree of automation in the public mirror application and approval process as well as whether newly listed mirrors have traffic immediately and automatically directed to them. In all cases, the distributions accepted our mirror and added it to the official mirror list for use by outside users. We saw traffic on our mirrors from a variety of clients, including military and government computers. More detail about our mirrors can be found in a tech report [5].

7.2 Security Repository

Debian and Ubuntu both use an official repository that serves security updates (packages that fix vulnerabilities). This prevents a mirror from launching a replay attack because the package manager will use the latest version of a package which will be available from the security repository. However, this protection does not extend to a man-in-the-middle attacker since the repositories do not support HTTPS.

Both Debian and Ubuntu use several mirrors beside the security repository. In the case where the security repository is down, an attacker can use a mirror to serve outdated content that was originally from the security repository to perform a replay attack. In addition, using multiple mirrors makes Ubuntu and Debian users much more vulnerable to endless data attacks.

7.3 Mirror Selection

In many distributions, not every source of data is created equal. For example, openSUSE distributes all of the metadata from a central source and only outsources package requests to mirrors. In this section we examine the impact that these practices have on the security of different distributions.

OpenSUSE uses a download redirector that sends some requests to mirrors. However, the download redirector serves package metadata and root metadata directly. This means that the client gets the metadata from a trusted source (not a mirror). While this doesn't protect against all of the attacks a client may face (such as endless data attacks by mirrors or replay attacks by a man-in-the-middle attacker), it does make it much more difficult to launch an attack.

On CentOS, clients contact a central service which redi-

rects the client's requests to official mirrors. Requests for all types of content (packages, package metadata, and root metadata) are directed to mirrors that are not controlled by CentOS. The specific mirror a client is redirected to may be different for each file requested. The result is that an average CentOS client contacts many more mirrors than would normally be the case. This makes it much easier for an attacker to conduct an endless data attack on many clients but complicates replay or freeze attacks because the same repository may not be contacted for packages and metadata. However, an attacker can still effectively launch an extraneous dependencies attack.

On Fedora, by default clients contact a central service called MirrorManager that is very similar to the CentOS service. However, `MirrorManager` allows a mirror administrators to specify that clients in an IP address range should use *only their mirror*. This allows easy targeting of attacks (to a specific country or organization) and reduces the number of other parties who will consume resources on the mirror. Note that the mirror need not have an IP address in the IP address range that it targets.

Online mirror redirectors handle failures differently than a security repository. If a mirror redirector fails, the clients will stop trying to communicate and fail (as opposed to "failing-open" by skipping the failed repository).

Many distributions including Gentoo, KNOPPIX, Arch Linux, Debian, and Ubuntu also have mirror selection tools that find a nearby mirror. These tools do not improve the security of users since they are only used to select a single mirror to download all content from. In fact, it should be noted that popular mirror selection tools for Debian and Ubuntu (such as netselect-apt or Software Sources) do not preserve the official security repository. This means that users who have used these tools to find faster or more reliable mirrors are unlikely to use the security repository for updates!

7.4 Verifying Mirrors

Many distributions use an automated mechanism to verify that mirrors are staying up-to-date. While we did not attempt to deviate from the correct, up-to-date status on our mirrors, we posit that an attacker can determine the IP address of the checking server and serve up-to-date content to the checking server while serving malicious content to users it is attacking. Note that this is trivial on distributions like Fedora that allow a mirror administrator to selectively target users, because the only references coming from outside the targeted range should be the checking server. It is publicly acknowledged by some distributions [13] that detecting and tricking these bots is trivial and so we do not feel this is a viable security mechanism.

7.5 HTTPS

Red Hat Enterprise Linux (RHEL) and SUSE Linux Enterprise are the only distributions that we found which widely support or use HTTPS. When investigating how YUM uses HTTPS, we discovered it does not validate SSL certificates. This means that while the communication uses HTTPS, there is no validation that the endpoint YUM communicates with is correct (allowing a man-in-the-middle to pose as the repository).

However, the Red Hat security team informed us that on RHEL, YUM communicates with the Red Hat Network servers using a special plug-in instead. During our discussion with the Red Hat security team, they realized there were issues in how SSL was used in the plug-in as well and began working on a fix for it.

7.6 Comparison

We examine the additional security mechanisms provided by different distributions and comment on their effectiveness against attack.

- SUSE Linux Enterprise doesn't use mirrors hosted by outside parties and uses HTTPS so it is not vulnerable to the attacks described in this paper.
- OpenSUSE, by using a download redirector, provides significant protection from malicious mirrors. However, any other attacker, such as a man-in-the-middle, that can respond to client requests can perform replay or freeze attacks. If the download redirector fails or is unreachable (e.g. due to a denial-of-service attack), users cannot get updates but are not at risk of replay attacks. Despite the download redirector protecting users from replay or freeze attacks from malicious mirrors, users are still vulnerable to endless data attacks from mirrors.
- Ubuntu and Debian have similar security to openSUSE with the additional problem that if the security repository fails, they become vulnerable to replay or freeze attacks from mirrors.
- Red Hat Enterprise Linux is not at risk from malicious mirrors (since it uses no mirrors hosted by outside parties) but is vulnerable to man-in-the-middle attackers because of flaws in their HTTPS implementation. A malicious party who can act as a man-in-the-middle can launch any of the attacks that YUM is vulnerable to.
- With a man-in-the-middle attacker, CentOS is vulnerable to every attack (much like RHEL). In the case of a malicious mirror, their download redirector makes it more difficult to launch attacks that require the attacker to provide a snapshot of packages (like a replay or freeze attack), but the potential for abuse from endless data and extraneous dependencies attacks is still large.
- Fedora has the same vulnerabilities as CentOS but with the additional problem that an attacker can target attacks to a specific IP range. This also allows an attacker to launch attacks that require the attacker to provide a snapshot of the repository.
- Other distributions such as Gentoo, Mandriva, LinuxMint, Sabayon Linux, Slackware, KNOPPIX, and MEPIS Linux do not significantly alter the protection provided by their package manager.

8. RESULTS

To understand the impact of an attacker that controls a mirror, a trace of package requests was conducted on a CentOS mirror and used to estimate the number of clients that could be compromised or crashed by an attacker. For all of the popular distributions our mirror could have launched

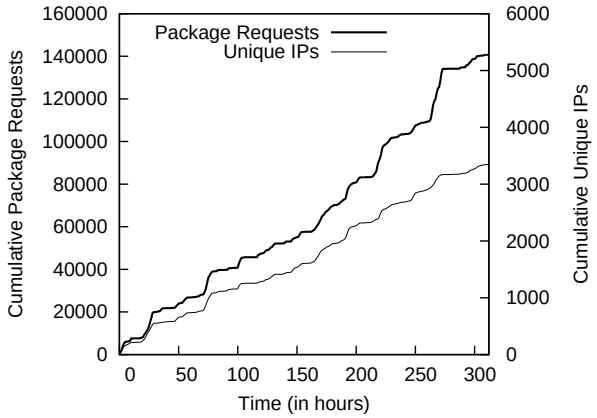

Figure 4: CentOS Mirror Traffic. This figure presents the cumulative package downloads and requests from unique IP addresses for the CentOS mirror over a 13 day period.

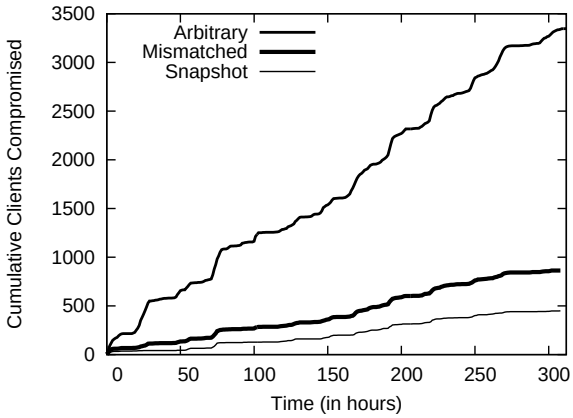

Figure 5: Compromised Clients CDF. This figure presents the cumulative number of clients compromised over a 13 day period for an attacker with 25 vulnerable versions. This figure shows the effect of the security mechanism on the number of clients compromised.

an endless data attack to crash all of the clients that visited it. The number of unique IP addresses that contacted our mirror represents an upper bound on the number of clients that could be crashed each week.

8.1 Mirror Traces

The CentOS mirror was chosen for this analysis because it was the longest mirror experiment that was conducted, lasting 13 days. The package access trace gathered from the CentOS mirror is shown in Figure 4. The number of package requests and number of requests from unique IP addresses increase roughly linearly over this time period. Assuming the CentOS user base is not growing faster than our mirror serves clients, we would expect the number of unique IP addresses to flatten out over time, however our trace is not long enough to capture that effect. Since clients are counted as unique by IP address, multiple clients behind a NAT box or proxy are counted as a single client. There are many instances where a single IP address has several orders of magnitude more package requests than the median client, often with many requests for the same package. This implies that clients are using NAT boxes or proxies in practice.

The openSUSE and Fedora mirrors (not shown) had similar traffic effects as the CentOS mirror. The Debian and Ubuntu mirrors (not shown) were both up for only a short period, but did not demonstrate this effect. We suspect this is because they do not automatically distribute requests among the mirrors, instead requiring manual selection by a user or the use of a tool like netselect-apt. Since our mirrors were only listed a few days, they did not attract a large number of Debian or Ubuntu users.

8.2 Package Versions and Vulnerability

To perform our analysis, we needed to know the distribution of package versions over time on the mirror, as well as which of those versions are vulnerable to attack. Information about the 58165 versions of the 3020 RPM CentOS packages used in the last year was captured. The update times were captured and used in the data set to determine if different versions existed on the main repository at the same time. This information was used to estimate compromises for those package managers that require a snapshot (packages that were all on a repository during the same time period).

Determining which package versions are remotely vulnerable to attack proved to be difficult and we are unaware of a data set that provides a good model of this. To compensate, we randomly chose a set of vulnerable versions from all non-current versions of packages. In practice, we believe that an attacker would be more likely to work to discover vulnerabilities in old versions of popular packages as these would allow the attacker to compromise more clients. This work does not capture this effect.

A client that uses a mirror with a package manager that has missing or inadequate security (Pacman, Slaktool, ports, YUM, and urpmi) is considered compromised whenever they install a package from the mirror. A client that uses a package manager that allows an attacker to mismatch vulnerable package versions (Portage) is considered compromised when it installs any package with a known vulnerability. For clients with snapshot vulnerability to replay and freeze attacks (Stork, APT, YaST, APT-RPM), the attacker chooses a snapshot time and can compromise clients who install a package that the attacker has a vulnerable version that was current at that time. We defer details of how the attacker chooses the snapshot time to a tech report [5].

Replay attacks only impact clients that are installing new packages instead of those that are updating existing packages. From the mirror's request log, it isn't possible to determine whether a client is installing or updating a package. This means the results for Portage, Stork, APT, APT-RPM, and YaST represent an upper bound. The package managers without security (Slaktool, ports, Pacman) are not impacted because an attacker can create arbitrary packages. Similarly, the results for YUM and urpmi are accurate because as long as there exists a vulnerable package the client hasn't updated to a newer version, an extraneous dependencies attack can compromise the client.

8.3 Number of Compromised Clients

Using the CentOS trace and version information, the number of clients compromised by a malicious mirror was estimated (Figure 5). Note that this estimation doesn't take into account the security practices of the distribution. As described in Section 7.6, users from distributions with additional security practices may be safer than what is described here. We provide information about the security of the package manager because many of the smaller distributions reuse package managers for more popular distributions and use only the security mechanisms provided by their package manager.

As the true number of vulnerable packages is not known, we use a value of 25 vulnerable packages (the effect of varying the number of vulnerable packages is shown in Figure 6). These plots show that the security model of the package manager has a great impact on the number of clients that can be compromised. A client that restricts attackers to mismatched vulnerable package versions reduces the maximum estimated number of compromised clients by about a factor of 4 to around 900 over the 13 day period. Package managers whose security mechanisms require a snapshot reduce the maximum estimated number of clients compromised to under 500. A package manager with signatures on the root metadata and protection against replay and freeze attacks (for example, modified Stork) will not have any compromises from an attacker that controls a mirror.

The package managers that allow the attacker to mismatch packages or that require a snapshot vary based upon the number of vulnerable packages. The effect of varying the number of vulnerabilities is shown in Figure 6. The number of packages with vulnerabilities that the attacker can exploit is on the x-axis. The plots show that package managers that allow an attacker to choose different versions of packages that existed on the root repository at different times (mismatch) are more vulnerable than package managers that require a snapshot.

This figure clearly shows that a package manager that requires an attacker to present a consistent set of packages provides better security than one that allows an attacker to mismatch packages. Somewhat disturbing is the significant number of clients that can be compromised if there are only 5 vulnerable package versions.

Our leased server was bandwidth-limited and mirrored multiple distributions simultaneously for cost reasons. An attacker would likely expend more bandwidth or set up multiple mirrors to capture additional traffic, thus leading to more compromises.

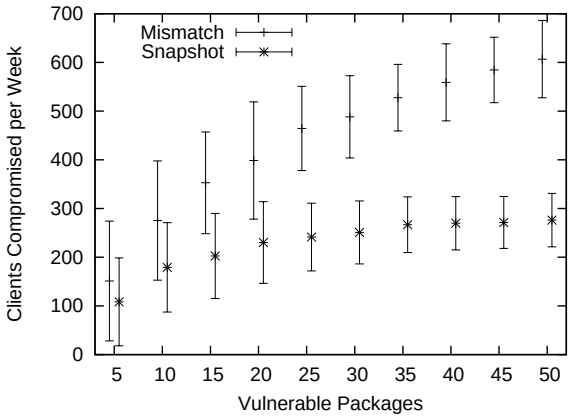

Figure 6: Clients Compromised Per Week. This figure presents the estimated number of clients compromised per week by an attacker that controls a mirror. The error bars show one standard deviation. The plots are offset slightly for readability.

9. DISCLOSURE AND DISCUSSION

After disclosing these vulnerabilities to package manager developers via Ryan Giobbi at CERT [12], we were contacted by proactive developers at Gentoo and openSUSE who confirmed our findings and began working on fixes [14]. We were also able to get in touch with developers at CentOS, Red Hat, Fedora, Ubuntu, and Debian and had confirmation and discussion about how we recommend they fix the issues we uncovered [10]. We believe that most of the other distributions will follow suit and either patch the vulnerabilities themselves or ask their package manager developer to do so. In the interim, we provide some suggestions about how distributions and users can increase their security:

- If the package manager supports HTTPS and it correctly checks certificates, the distribution can set up repositories or mirrors that support HTTPS transfers. This will not protect against a malicious mirror, but will prevent a man-in-the-middle attack from an outsider.
- The distribution can review their mirror policy carefully and validate administrator credentials before putting a mirror on the official mirror list. The distribution should also review their policies for deciding how mirrors are allocated traffic to prevent a mirror from being able to target attacks (perhaps indirectly by advertising high bandwidth).
- The distribution's mirrors should use a secure connection (e.g. SSH) to synchronize with the main repository to prevent an attacker from impersonating the main repository.
- Users should check that the versions of the packages their package manager recommends for installation are recent through multiple sources.

10. RELATED WORK

Many package managers have GUI front-ends [15, 27]. These GUI-based tools are usually just a different interface to the functionality provided by a command line package manager and so are identical from a security standpoint.

There are many techniques that help to support software security such as systems that ensure the authenticity and integrity of software (including SFS-RO [16], SUNDR [17], Deployme [20], and Self-Signed Executables [30]), and code signing certificates [6]. These are complimentary to the solutions presented in this paper.

There are several systems that access multiple mirrors to improve download performance and avoid DoS attacks. Byers et al. [4] describe using Tornado codes to improve performance by downloading from several mirrors simultaneously. This has the side-effect of allowing the client to make progress even if one of the mirror misbehaves. Sharma et al. [24] describe having the client "hop" between mirrors

while downloading a file. This prevents an attacker from launching a DoS attack on the client because the attacker does not know which mirror the client will use next. We hope that raised awareness will result in organizations trying these techniques in practice.

11. CONCLUSION

This work identifies security issues in ten popular package managers in use today. Furthermore, we demonstrate that while some security mechanisms used by distributions may help to prevent attacks, others may actually decrease security. We estimate that when ignoring any additional security mechanisms, an attacker with a mirror that costs $50 per week can compromise between 150 and 1500 clients each week. These security issues have been disclosed to distributions that are currently working to fix these problems.

12. ACKNOWLEDGMENTS

We would like to thank our shepherd Steven Murdoch and the anonymous reviewers for their insightful comments and feedback. We would like to thank Ryan Giobbi at the CERT for helping us to responsibly disclose these issues to the package manager maintainers. We would like to thank Jake Edge, Dag Wieers, Kees Cook, Jan iankko Lieskovsky, Robin Johnson, Ludwig Nussel, Peter Poeml, Marcus Meissner, Josh Bressers, Tomas Hoger, Jason Gunthorpe, Kyricos Pavlou, Mike Piatek, Chris Gniady, Wenjun Hu, and Tadayoshi Kohno for their comments on our research. We would also like to thank the members of the Stork project for their help in creating Stork and their input on this research. This work was funded in part by the PlanetLab Consortium.

13. REFERENCES

[1] Debian APT tool ported to Red Hat Linux. http://www.apt-get.org/.

[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] J. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: using Tornado codes tospeed up downloads. *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 1, 1999.

[5] J. Cappos, J. Samuel, S. Baker, and J. Hartman. A Look In the Mirror: Attacks on Package Managers. Technical Report TR–08–06, Department of Computer Science, University of Arizona, Jul 2008.

[6] Introduction to Code Signing. http://msdn2.microsoft.com/en-us/library/ms537361.aspx.

[7] A. Crooks. The netbsd update system. In *ATEC '04: Proceedings of the USENIX Annual Technical Conference*, pages 17–17, Berkeley, CA, USA, 2004. USENIX Association.

[8] debsigs - What is debsigs. http://linux.about.com/cs/linux101/g/debsigs.htm.

[9] DistroWatch.com: Editorial: How Popular is a Distribution? http://distrowatch.com/weekly.php?issue=20070827#feature.

[10] M. Domsch. Re: YUM security issues... https://www.redhat.com/archives/fedora-infrastructure-list/2008-July/msg00114.html.

[11] man dpkg-sig. http://pwet.fr/man/linux/commandes/dpkg_sig.

[12] R. Giobbi. Vulnerability Analysis Blog: Safely Using Package Managers. http://www.cert.org/blogs/vuls/2008/07/using_package_managers.html.

[13] J. Hughes. HughesJR.com – Attacks on Package Managers – ummm... http://www.hughesjr.com/content/view/22/1/.

[14] R. H. Johnson. [gentoo] Index of /users/robbat2/tree-signing-gleps. http://viewcvs.gentoo.org/viewcvs.py/gentoo/users/robbat2/tree-signing%-gleps/.

[15] The KPackage Handbook. http://docs.kde.org/development/en/kdeadmin/kpackage/.

[16] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proc. 17th SOSP*, pages 124–139, Kiawah Island Resort, SC, Dec 1999.

[17] D. Mazières and D. Shasha. Building secure file systems out of Byzantine storage. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 108–117, New York, NY, USA, 2002. ACM.

[18] milw0rm - exploits : vulnerabilities : videos : papers : shellcode. http://www.milw0rm.com.

[19] Netcraft: Strong growth for Debian. http://news.netcraft.com/archives/2005/12/05/strong_growth_for_debian.%html.

[20] K. Oppenheim and P. McCormick. Deployme: Tellme's Package Management and Deployment System. In *Proc. 14th Systems Administration Conference (LISA '00)*, pages 187–196, New Orleans, LA, Dec 2000.

[21] Gentoo-Portage. http://gentoo-portage.com/.

[22] Installing Applications: Packages and Ports. http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ports.html.

[23] RPM Package Manager. http://www.rpm.org/.

[24] P. Sharma, P. Shah, and S. Bhattacharya. Mirror hopping approach for selective denial of service prevention. *Object-Oriented Real-Time Dependable Systems, 2003.(WORDS 2003). Proceedings of the Eighth International Workshop on*, pages 200–208, 2003.

[25] Slackware Package Management. http://www.slacksite.com/slackware/packages.html.

[26] Stork. http://www.cs.arizona.edu/stork.

[27] Synaptic Package Manager - Home. http://www.nongnu.org/synaptic/.

[28] URPMI. http://www.urpmi.org/.

[29] dkpg-sig support wanted? http://nixforums.org/about101637-asc-15.html.

[30] G. Wurster and P. van Oorschot. Self-Signed Executables: Restricting Replacement of Program Binaries by Malware. In *2nd USENIX Workshop on Hot Topics in Security*, Boston, MA, Aug 2007.

[31] YaST - openSuSE. http://en.opensuse.org/YaST.

[32] Yum: Yellow Dog Updater Modified. http://linux.duke.edu/projects/yum/.