A Brief Security Analysis of Arch Linux and its Package Management System

Brandon Milton*

June 11, 2017

Abstract

Arch Linux is an independently developed GNU/Linux distribution which has remained popular for personal computers for the past decade. Without the support of a company or organization that many other popular distributions have, it might be expected that Arch Linux lags behind in terms of security. In this paper, I analyze the trust model of Arch Linux and its underlying implementation. I also analyze the security of Arch Linux's package management system by applying common attacks and analyzing code. I find that the Arch Linux community cares about security and employs a strong trust model, but touts a package management system that is susceptible to common attacks.

*University of California, San Diego - bmilton@ucsd.edu

Contents

1 Introduction

Arch Linux is an independently developed and community-maintained x86-64 GNU/Linux distribution which has steadily been gaining popularity since its initial release in 2002. According to `distrowatch.com`, Arch Linux has been a top-10 choice for personal computers since 2009.[1] The distribution prides itself in its simplicity and user-centrality, with a focus on helping current users over attracting new ones.

As an operating system, Arch Linux does not do anything special. Much like other GNU/Linux distributions, Arch Linux is a collection of software bundled with the Linux kernel. Of course, Arch Linux does have its own software bundled with the operating system, most notably its package manager and its installation scripts.[2] On Internet forums, however, Arch Linux stands out as an operating system much different from other GNU/Linux distributions. It has garnered a reputation for being both difficult to use and not-so-stable. The former is mostly due to Arch Linux's lack of default graphical user interface. While most distributions come bundled with a graphical installer and a default desktop environment, Arch Linux is installed via a terminal emulator and forces the user to select their own graphical tools, if any at all.[3] The latter is a byproduct of the community's desire to always have only the most up-to-date software on their systems. Because of this, software which depends on older versions of other software often becomes broken. Although small, both of these differentiators have interesting security implications.

The primary differentiating factor between Arch Linux and other distributions, however, comes in community. The distribution prides itself on community involvement and offers many ways for individuals of the community to aid in the development and operation of the system. The ArchWiki, for instance, is a community maintained wiki and serves as the primary source of information on anything relating to Arch Linux. As another example, the Arch User Repository is a collection of roughly 43,000 community-maintained packages that users may download to their systems using tools they already have. This large amount of community involvement also has interesting security implications.

Of all the distributions on `distrowatch.com`'s top 10, Arch Linux is the only distribution that is developed and maintained without the support of a company or non-profit organization.[4] Not only does this make Arch Linux unique, but it also creates interesting models of trust, privacy, and security.

In this paper, I intend to explore various parts of Arch Linux security. I will analyze the trust model of Arch Linux and the tools it uses to accomplish this trust model. Then, I will analyze the security of Arch Linux's package management system, `pacman`, and the Arch User Repository. Finally, I will analyze how important security and privacy are to the users of Arch Linux by analyzing the attitudes and instructions of mailing list correspondence.

With this paper, I intend to answer the question of whether Arch Linux takes security seriously.

2 Downloading Arch Linux

The first step in using Arch Linux is downloading its installation image. Arch Linux hosts a directory of mirrors to download the official image, as well as BitTorrent links, on its official website.[5] At the time of writing, there are 338 different mirrors around the globe hosting the Arch Linux installation image. For integrity and authenticity purposes, each mirror also hosts copies of SHA1 and MD5 hashes of the image as well as a RSA-3072 OpenPGP signature of the image.[6] Since it's entirely possible that a mirror may be malicious, the official Arch Linux downloads page also hosts the same hashes and signature files. Thus, Arch Linux avoids having to host the large image file for its users, while also providing the users a trusted way to verify the authenticity of the download.

Of course, MD5 and SHA1 are no longer considered secure. Although they may have limited purpose in quickly verifying the image file's completeness, providing them does not add any extra security. Indeed,

[1] Of course, there is no formal way of measuring distribution market share, but `distrowatch.com` is a decent approximation.

[2] For a complete list of Arch Linux software, consult the official list of git repositories at `https://git.archlinux.org/`

[3] Because of how daunting installation via commandline can be, there are entire tools dedicated to providing an easier install experience for Arch Linux (Arch Linux Anywhere, Revenge Installer). Distributions based off of Arch Linux such as Antergros and Manjaro also have graphical installers

[4] Of the top 10, Debian is supported by its corporate partners; Ubuntu is based on Debian and supported by its developing company, Canonical; 3 are based on Ubuntu; 2 are supported by Red Hat; openSUSE is supported by SUSE. Two distributions are not supported by an organization, namely Arch Linux and Manjaro Linux, which is based on Arch Linux.

[5] `https://www.archlinux.org/download/`

[6] PGP, short for "Pretty Good Privacy", is a trademark of the PGP corporation. Arch Linux, however, uses the open source variant known as OpenPGP. All references to "PGP" throughout this paper will refer to OpenPGP.

if a malicious mirror also somehow hacked the Arch Linux site, they could also change the hashes. The PGP signature, however, being bound to a trusted Arch Linux mantainer's RSA-3072 private key, does an ample job of protecting against possibly malicious mirrors. Even in the case of a website hack, the authenticity of the installation image could only be confirmed if the PGP signature came from a trusted public key. The unsigned MD5 and SHA1 hashes, therefore, seem to offer nothing in terms of security, and only serve as a way to verify the downloaded image file. Authenticity may only securely be confirmed via the PGP signature.

Further, of the 338 websites hosting the installation image, only 89 are protected by HTTPS. For those not using HTTPS, it is possible that the download is altered by a man-in-the-middle attack. Once again, however, properly verifying the download against the officially listed hashes or the PGP signature would reveal this.

In practice, it is unclear how many users actually verify the image download. However, those who may be new to the process are indeed advised to verify the PGP signature in the official installation guide.[7]

Thus, the security of downloading Arch Linux ultimately relies on the security guarantees that PGP provides, the trust placed in the signer of the installation image, and the ability for the signer to keep his private key secret.

3 The Trust Model

The trust model of Arch Linux is an interesting one. With most distributions, the trust model is relatively simple. Usually, there is a single entity that the user may place trust in, be it a company or organization. Therefore, anything received from this company or from someone trusted by this company should also be trusted by the users. For a user, a company may be easy to trust. A company is easily recognizable, easy to identify with reviews or previous bad experiences, and easy to hold accountable.

Since Arch Linux is independently developed and community maintained, it becomes unclear who the users should trust. The Arch Linux community is built up of many loosely connected individuals, and there is no single entity that defines the community. Furthermore, anyone can be part of this community and people may join or leave at any time. Giving authority to some members of the community, however, offers a solution to this problem.

Arch Linux has chosen to give 5 community members authority over all others.[8] All users, then, are assumed to at least marginally trust these 5 community members. Anyone who is trusted by a majority of these 5 members should then be trusted by the users. In order to handle the case when one of the 5 trusted members becomes malicious, each of the 5 members is also assigned a "revoker", who holds the member's revocation certificate and has the power to invalidate the member's keys.

Arch Linux uses public key encryption, specifically the OpenPGP protocol, to handle this. The private keys of the 5 trusted members are known as "master signing keys".[9] By default, when a user installs Arch Linux, the public keys of these 5 members are loaded into the system, signed with the system's key, and marginally trusted. Along with the 5 master keys, a new installation also includes the public keys of Arch Linux developers and trusted users. Following OpenPGP's web-of-trust model, the keys of these individuals are considered valid if they are signed by a majority of the 5 master keys.[10] Furthermore, this is true for any other available public key: it is considered valid if it is signed by a majority of the master keys. In short, all sense of trust in Arch Linux originates from assumed marginal trust in the master keys.

4 The Security of PGP

Arch Linux relies directly on OpenPGP to create and carry out its trust model. Thus, the security of Arch Linux's trust system is intertwined with the security of the OpenPGP protocol. Similarly, any shortcomings in the OpenPGP protocol may also imply shortcomings in the security of Arch Linux.

[7] `https://wiki.archlinux.org/index.php/Category:Getting\_and\_installing\_Arch`

[8] This number is not set, and the number of "master users" may change at any time

[9] The current Master Signing Key arrangement can be viewed on the official Arch Linux website at `https://www.archlinux.org/master-keys`

[10] Do note that, by default, the individual keys are considered authentic if they are signed by 3 marginally trusted keys. For Arch Linux, this simply happens to be the majority of master keys. As the number of master keys grows, the "majority" requirement may no longer hold.

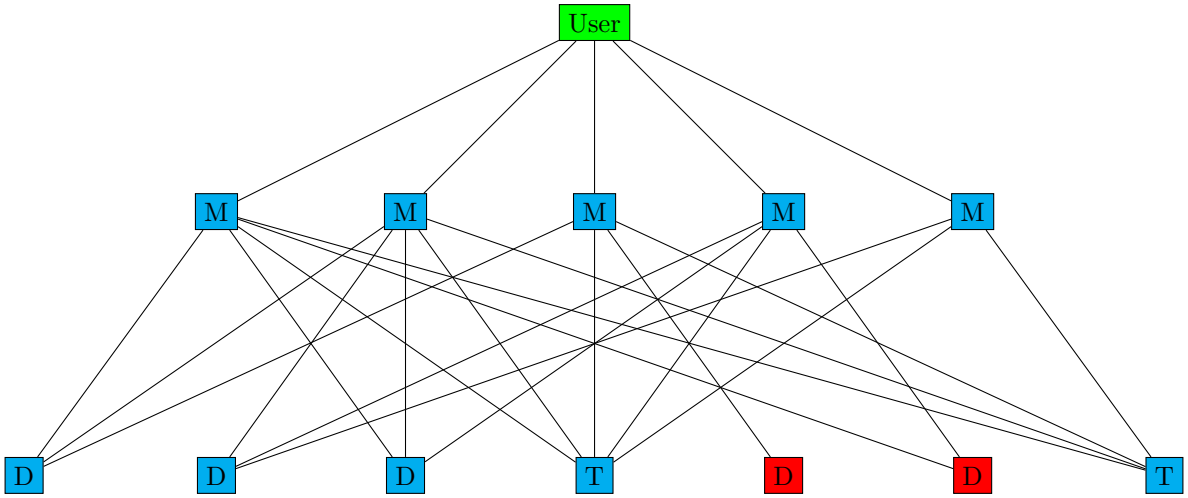

Figure 1: A graphical representation of the Arch Linux trust model. M nodes represent master keys, D nodes represent developers, T nodes represent trusted users. An edge represents a signature, where the higher node has signed the lower node's public key with their own private key. Green nodes are considered "ultimately" trusted. Cyan nodes are considered fully valid from the user's perspective, while red nodes are not considered fully valid from the user's perspective.

The most common criticism of PGP is its lack of forward secrecy.[2] That is, in the event of a key compromise, both all past and future communications are compromised. This is due to PGP's use of single, long-term keys, for communications. Arch Linux, however, only uses PGP as a form of authentication. This criticism, which applies when using PGP as a measure of privacy, does not apply to Arch Linux directly. However, the usage of single long-term keys for package signing does result in some interesting problems for Arch Linux, which will be discussed later.

Although not inherent to the protocol itself, modern usage of PGP relies on keyservers - large databases of public keys searchable by key fingerprint or other identifiers such as email address or name. Although most PGP clients can only search keys by their fingerprints, some keyservers have a user interface which allows the user to search for public keys by name or email. In this case, it is possible that a PGP user may import, sign, or trust the wrong key. Although the PGP protocol implies that users should verify keys before signing, it is possible that a user may trust the name to fingerprint mapping that is displayed by the keyserver UI. In terms of Arch Linux, an attacker could impersonate a trusted Arch Linux user, whose names are publicly available.[11] A user may then mistakenly find and trust the malicious key, making it possible for the attacker to deliver badly-signed binaries to this user.

Several other criticisms of PGP revolve around the Web of Trust.[6] Although many of the criticisms are valid, some are not particularly pertinent to Arch Linux's use of PGP.

The complaint that the Web of Trust leaks information, for example, is a valid one, but usually only applies when using PGP as a communication mechanism or when publicly signing other keys. For normal Arch Linux users, this isn't a problem. Since PGP trust levels are managed via the `archlinux-keyring` package, a normal user should never have to sign a PGP public key. Further, Arch Linux automatically creates a new private key upon installation of the system, using its `pacman-key` tool and only locally signs master keys with this key. Thus, Arch Linux's use of PGP is not conducive to Web of Trust information leaks. Outside of normal operation, a user might want to sign an Arch Linux developer key, which would at most leak the user's usage of Arch Linux as well as their favorite Arch Linux developer. In some cases, the AUR's workflow is more conducive to information leaks, which will be discussed later.

Complaints about Web of Trust scalability are also warranted, but inapplicable to the majority of Arch Linux users. In the Arch Linux ecosystem, there is at most one trust-hop from a Master Key to a Trusted Developer Key. Further, all of this trust is managed via the `archlinux-keyring` package. Thus, there is no need for users to meet with others in order to exchange official Arch Linux keys. At the time of writing, there are 68 developer keys signed by master keys [12], also making storage of keys very manageable.

[11] `https://www.archlinux.org/master-keys/`

[12] Information about master key signatures can be found at `https://www.archlinux.org/master-keys`

A final common complaint about the Web of Trust is that it allows for single points of failure.[13] A single, fully trusted entity has the power to certify any other keys, some of which might be malicious. This situation could seemingly pose serious problems in the Arch Linux community, where a single master key could, by no fault of his own, sign the key of an individual who uploads malicious packages. This situation is only scary if users fully-trust this Arch Linux master key. Arch Linux solves this problem by having 5 master keys, each one being only marginally trusted by users. In this way, other keys are only considered authentic by users if they are signed by 3 of the 5 master keys. Thus, in order for the master keys to act as a failure point, several would have to collude. Even so, these 3 master keys would have to escape the watching eye of their revoker, who might choose to invalidate the signatures of one master key. Of course, a situation where a master key is colluding with his revoker would prevent invalidation.

5 Description of `pacman`, the Package Management System

Package Management is an extremely important aspect of an operating system, especially a Linux Distribution. The package manager is the primary interface for users wishing to install new software or update to the latest versions. Most package management systems are also curated by the operating system developers, meaning that packages are only added to the package management system once they are verified to be compatible with the systems.[4]

Since users use the package manager to manage system software, it is usually run with elevated privileges. Further, users rely on package managers to provide security patches to software vulnerabilities. Therefore, it is extremely crucial for a package management system to be secure. For Arch Linux, the package manager also manages core system updates. In Arch Linux's "rolling release" model, all core system and kernel updates are provided via the package manager.

Running a package management system also involves maintaining databases of package information, hosting the databases and their binaries, and keeping an up-to-date list of hosts that the user may access. Arch Linux, being a small and independent distribution, relies on "mirrors" to do the hosting for them. All these factors combined create a perfect system for an adversary to exploit.

5.1 Mirror Layout

In order to propagate upstream packages throughout the package management system, Arch Linux has designed a tiered mirror system.[14] That is, all updates to the package databases and package binaries (as well as a "last changed" timestamp) are published to `archlinux.org`. These updates are only downloadable by Tier 1 Mirrors, who in turn provide these files to both users and Tier 2 Mirrors. Tier 2 Mirrors sync from Tier 1 mirrors, but only provide files to end-users.

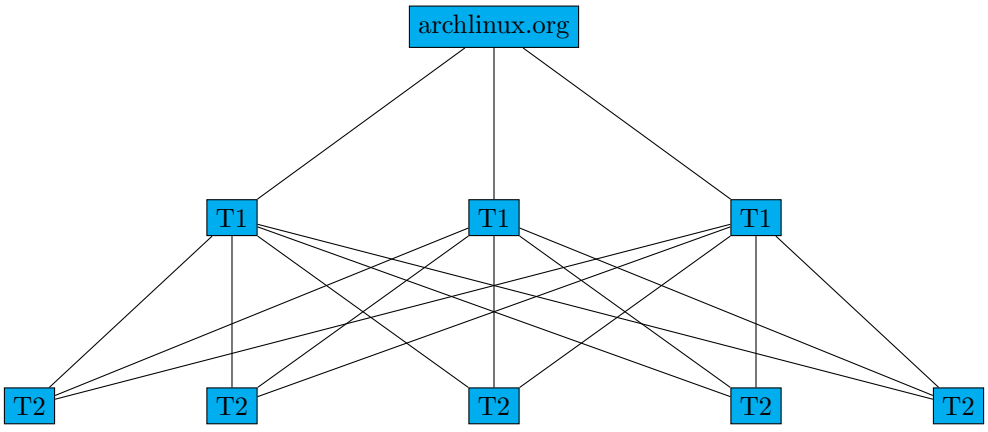

Figure 2: The syncing layout of Arch Linux's package management system, where $T1$ nodes represent Tier 1 mirrors and $T2$ nodes represent Tier 2 mirrors

[13] Although this is the final PGP complaint that I will analyze in this paper, there are many more valid PGP complaints found around the web. There are some articles, like the one found at `http://secushare.org/PGP`, which advocate for never using PGP at all.

[14] More information is available on the official ArchWiki page `https://wiki.archlinux.org/index.php/DeveloperWiki:NewMirrors`

When a new Tier 1 or Tier 2 mirror is added, the URL of the mirror is added to the official mirrorlist[15] and is eventually propagated to users via an update to the `pacman-mirrorlist` package. Users may also choose to manually populate their mirrorlist, choosing to sort by speed or protocol preference, or use tools to manage this.[16]

Since Arch Linux relies on third-parties to maintain its package-management infrastructure, it almost seems inevitable that a malicious mirror take advantage.

5.2 Package and Database Layout

It is the responsibility of the aforementioned mirrors to host both the package databases and the package binaries. Packages are further split into repositories according to their function. Tools core to Arch Linux functionality and development are put into the `Core` repository; Software that is useful to have is put into the `Extra` repository. Both `Core` and `Extra` are managed by "Trusted Developers". Software that is useful, but not a priority of primary Arch Linux developers is put into the `Community` repository and is managed by "Trusted Users". There exists a database for each repository, which holds metadata about all packages in the repository, their versions, maintainers, hashes, and signatures. For simplicity, the database is simply a compressed collection of text-files, one per package, which hold key-value pairs of all relevant metadata.[17] Although the tooling officially supports package database signing, there are no mirrors which host databases that are signed by a trusted Arch Linux developer. Further, there are no means of validation or authentication for the package database.

Package binaries are also hosted by the mirrors. The binaries, as well as select metadata about the binaries, are bundled, compressed, and stored in files named from the package name and version. These bundles are also signed by "Trusted Developers" (or "Trusted Users" in the case of the `Community` repository) using their PGP private keys. These signatures are stored in the database file as well as in a separate file stored alongside the binary package.

In order to prevent mirrors from serving out-of-date content, Arch Linux maintains a *lastsync* file, which contains the timestamp of the last repository update. After syncing their repositories, mirrors will also receive the updated *lastsync* file and can thus be checked for staleness. An automated tool compares mirrors' *lastsync* to the master *lastsync* and marks any non-matching mirrors as out-of-date.

When downloading packages, users reference their locally downloaded copies of each repository's database, using the database's version information to retrieve the package and the database's signature information to validate and authenticate the download. When a user's database becomes stale (a database is considered out-of-date if its file modification time is earlier than that of the mirror's database), they may update it by downloading a new copy directly from a mirror. Thus, in order for a user to get a newly released version of a package, they must first update their database to learn the new version number, then download the package.

6 Security of `pacman`, the Package Management System

The threat model of Arch Linux's package management system involves either a man-in-the-middle attacker or a malicious mirror. Although some other forms of attack might exist, we assume that the trusted individual who has built, signed, and released the package took proper means to prevent other attacks.

A man-in-the-middle attack is not very hard to conceive. At the time of writing, there are 254 mirrors, 34 of which are Tier 1, 220 are Tier 2. All 34 Tier 1 mirrors offer communication on an unencrypted protocol (either HTTP or rsync), while only 19 offer communication along HTTPS. 211 Tier 2 mirrors offer unencrypted communication, while only 70 offer HTTPS. Thus, it is entirely conceivable that a MITM might intercept-and-replace traffic from a Tier1 to Tier2 mirror or from a user to a mirror. In either case, however, any tampered packages would be detected by the end-user since they are PGP signed. Package databases, however, remain unsigned and are thus vulnerable to a MITM attack on select requests on select mirrors. The level of security that a user receives against MITM attacks is ultimately up to them. By default, a list of all mirrors are bundled with an Arch Linux installation. However, a user may choose to only retrieve data from mirrors serving over HTTPS.

[15] The official mirrorlist can be viewed at `https://www.archlinux.org/mirrorlist/all/`

[16] A popular tool for managing the Arch Linux mirrorlist is `Reflector`, which can be found at `https://wiki.archlinux.org/index.php/reflector`

[17] Package groups have special entries in the package database, but I will not be analyzing them.

The Arch Linux package management system is also conducive to malicious mirror attacks. As discussed by Cappos et. al, there is no reason that a given mirror must be benevolent, and there are several attacks that this malicious mirror might be able to perform. Further, due to Arch Linux's tiered-mirror system, a non-benevolent mirror has the potential to affect both users and other mirrors. I will now analyze how some attacks described in the work by Cappos et. al apply to Arch Linux.

6.1 Package Database Alteration

An easy way for malicious mirrors to perform attacks is by altering package metadata found in the package database. Of course, an easy way to prevent this tampering is to cryptographically sign the databases and confirm their authenticity upon receipt. Unfortunately, although Arch Linux's package management system has the capability to perform database signature checking, there are no mirrors hosting signed package databases. Thus, delivering an altered package database to an end-user is trivially easy in Arch Linux. Further, an end-user accepts any database file which has a last-modified timestamp greater than their own. Therefore, a mirror may ensure that a user always receives an altered database by always updating the database file's modification time.

Since the package databases are originally published on `archlinux.org` and propagated to mirrors, it is entirely possible for automated tools to detect when mirrors are "out-of-sync" and serving bad data to users. Indeed, Arch Linux employs an automated tool to check the validity of all mirrors. However, they do so only by verifying that the mirror's last update timestamp is equivalent to `archlinux.org`'s last update. It is entirely possible that a mirror host bogus data while keeping their *lastupdate* file up-to-date. Further, since these checks are performed on a routine basis (every 30 minutes)[18], a mirror is able to detect from which machines the validity-checks originate and selectively serve valid data only to those machines. Thus, a mirror might continue serving bad package databases while never being marked as "out-of-sync". A better solution might involve a randomly-intervaled tool comparing the hash of a mirror's database file to the hash of the latest version. However, a malicious mirror could still filter requests based on source address.

A mirror might also maliciously choose to pin the database version, constantly providing users with a stale version of the database. This would keep users' packages at a specific version, meaning that the mirror could target these users when package vulnerabilities are announced. In Arch Linux, this problem is a little less likely. In an effort to always have only the most up-to-date packages, mirrors only keep copies of up-to-date binaries. Thus, when a user being kept on an old database makes package requests to other mirrors, the requests would fail, since no other mirror would be hosting older versions of the packages. Thus, this attack is only truly successful if users visit the same malicious mirror for both database and package updates.

Once a mirror has been marked as "out-of-sync", there is still an issue of removing the mirror from users' mirrorlists. In Arch Linux, these mirrorlist updates are maintained in the `archlinux-mirrorlist` package. When a user receives an update to this package, they must then manually apply the updated mirrorlist (to prevent overwriting of user mirror preferences). This extra manual step prevents "out-of-sync" updates from immediately propagating to users. Further, it is entirely possible for a mirror to "pin" an old version of `archlinux-mirrorlist` so that a user never receives these updates. The package management system could fix this issue by publishing their trusted hash of the latest database and validating user-downloaded databases against this hash. In addition, the package manager could download databases from multiple mirrors and confirm that all hashes agree before accepting the new database. As a user, these problems might be prevented by regularly randomizing the local mirrorlist such that a different mirror is used for every database update. A user may also use a automated mirrorlist tool, minimizing reliance on the `archlinux-mirrorlist` package.

The majority of database-alteration problems are indeed solved by database signing. Fortunately, the infrastructure is in place to enable database signing, signed databases simply have to begin being hosted. In order to prevent database "out-of-sync"ness, Arch Linux may choose to implement either hash-comparisons of databases or expiration dates on database signatures. For a more robust solution (but with a negative impact on speed), the package management tool could also query multiple mirrors for databases, compare hashes, and disregard any outliers.

As a user, the threat is mitigated by using only trusted mirrors and by regularly applying updates to the mirrorlist. For added safety, the user could also regularly randomize their mirrorlist, making sure that they do not use a single mirrorlist for repeated updates. Unfortunately, using only trusted mirrors

[18] A brief summary of how often Arch Linux checks mirrors for stale data can be seen on the official mirror status page, found at `https://www.archlinux.org/mirrors/status/`

is also not a complete solution if package databases are unsigned. Due to Arch Linux's tiered-mirror system, it is entirely possible that a malicious Tier 1 mirror begins serving altered package databases to trusted Tier 2 mirrors, which eventually propagate to the user.

6.2 Old Package Versions

As described by Cappos et. al, the primary attack vector of a a malicious mirror is to prevent the user from applying security upgrades and instead provide the user with a older version of a package, one with a security vulnerability.

In the current state of Arch Linux, is it extremely easy to provide the user with an older version of a package. Since the package database is unsigned, the malicious mirror simply has to change the version number, signature, and size entries of a given package. Then, upon downloading the database and requesting the package in question, the user would request an older version.

As previously mentioned, this is a little difficult since Arch Linux mirrors usually only host the latest versions of all packages. Thus, the malicious mirror would also have to host the older version with the security vulnerability. Of course, this version was once signed by a "Trusted Developer" and is still valid. The mirror would then have to hope that the user visits it for both package metadata and binaries in order to avoid detection and launch a successful attack.

As pointed out by Cappos et al, even when databases are signed, the malicious mirror can simply use an old, signed, copy of the database to provide the old version number for the package.

Arch Linux's package management system does take small steps to prevent this. When performing a system upgrade, the user has to pass an additional flag in order to specify that packages should also be downgraded. Further, when packages are being downgraded, a warning message is displayed. Upon being downloaded, the version specified in the database is compared to that specified in the signed binary. If they do not match, the installation does not proceed. Thus, malicious mirrors must alter the database if they wish to provide old packages to users.

6.3 Dependency Alteration

Since the package databases remain unsigned in Arch Linux, it is also possible to alter the dependencies of packages. Not only is it possible to make all packages depend on a vulnerable package, but it is also possible to prevent users from downloading a package by making it depend on a package that does not exist.[3]

In Arch Linux's case, there is information about dependencies stored in both the package database as well as the signed binary package. Therefore, it is possible to prevent unwanted dependency installation by comparing the database dependencies to the signed binary package's dependencies; however, this comparison is simply not performed.

6.4 Infinite Data

When the user requests data from a mirror, it seems plausible that a malicious mirror might send an infinite stream of data to hog precious user resources.[3] In Arch Linux, there is a declared upper limit on the size of package databases. While downloading, if the size exceeds this upper limit, the download is stopped, thus preventing infinite data while downloading databases.

The filesize for packages, however, is declared in the package database (in bytes). Since package databases are unsigned, a malicious mirror may edit the file size for a package to be the upper limit of its datatype, in this case a signed integer. Thus, a malicious mirror might not be able to send infinite data, but a large amount. However, before agreeing to download a package, `pacman` displays the filesize for the package, making the user well aware of the mirror's intentions.

6.5 Verdict on the Security of `pacman`

In its current state, `pacman` is extremely susceptible to both MITM and malicious mirror attacks. Although there are security mechanisms that are supported, such as package signing and database/package comparisons, they are simply not utilized to increase the security of `pacman`. However, the amount of textual warnings `pacman` displays to the user during a possible attack is notable. Perhaps it is Arch Linux's goal to simply make the end-user aware of the problems so that they may report it, making the damage short-lived.

Unfortunately, if an end-user were to detect something wrong, there is no official way to report a malicious mirror or ongoing MITM attack. Perhaps a simple "report" button on the mirror status page would allow more users to report malicious intent. Arch Linux may have the view, however, that those who might notice a malicious act should also know how to use a mailing list.

For an independently organized distribution, `pacman` does well. However, it still is susceptible to many attacks described elsewhere[3], where known remedies exist.

7 AUR: The Arch User Repository

7.1 What is the AUR?

The *Arch User Repository*[19] is a collection of user-uploaded and user-maintained packages. Contrary to the `Core`, `Extra`, `Multilib`, and `Community` repositories, which hold binary packages uploaded by trusted users, the AUR holds source packages uploaded by non-trusted users. An end-user downloads a source package from the AUR, verifies the source code, and uses the `makepkg` Arch Linux system tool to run the included build script (called a *PKGBUILD* file), which compiles and creates a package installable by `pacman`, Arch Linux's package manager. The user can then install the package. The AUR allows developers to upload their own software into the Arch Linux ecosystem, but also allows users to upload other popular software (such as Skype or Spotify), which would otherwise not be available as an official Arch Linux package. Occasionally, a package hosted in the AUR might be sponsored by a "Trusted User" and promoted to the `Community` repository.[20] The AUR is also monitored by "Trusted Users".[21]

The AUR, by convention, is not meant to host trusted packages nor secure ones. In fact, the homepage of AUR sports the disclaimer "AUR packages are user produced content. Any use of the provided files is at your own risk." Tools which automate the process of interacting with the AUR, although helpful, are usually discouraged since they release the end-user's responsibility of manually verifying source code and *PKGBUILD* files.

It may not be meant to be secure, but the AUR remains a very popular aspect of Arch Linux. Thus, it still remains interesting to analyze the trust models and security of the AUR.

7.2 Trust in the AUR

The trust model in the AUR is fairly simple. Excluding the developers of the source code, an end-user must only trust the maintainer of the AUR package. There is no minimum requirement of trust for publishers of AUR packages. If he chooses to do so, the publisher may omit any identifying information, except the username he uses to publish the package. In this case, unless the end-user trusts the publisher's username, it is entirely up to the end-user to manually audit both the *PKGBUILD* file and the source-code of the program. I refer to this mode of *PKGBUILD* operation as "trustless".

In order to provide means for authentication, the publisher of the package may include a PGP signature of the package source code in the *PKGBUILD* file. Once this is included, when the end-user runs `makepkg`, the PGP signature will be validated using the end-user's personal keyring.[5] If this validation fails, the source code will not be compiled and the process will be aborted. In the case that the end-user does not have the publisher's public key, he must retrieve it using standard PGP key-retrieval methods.[22] Now, whether the end-user trusts the source code operates entirely under PGP, where the end-user will trust the public key only if he signs the public key directly, or it is signed by someone trusted by the end-user. I refer to this mode of *PKGBUILD* operation as "trust-enforced".

This "trust-enforced" mode does have a harmful side-effect, however. In an effort to get necessary software onto their machine, the end-user may be unwilling to fully obtain trust information about the publisher's key. In order to quickly install software, the end-user may blindly sign the publisher's key for the duration of the installation process. However, if the user does not revoke their signature after the installation process, they are left with a signed, public key on their personal keyring. Any future PGP

[19] The AUR homepage is found at `https://aur.archlinux.org`. The relevant ArchWiki entry for the AUR is found at `https://wiki.archlinux.org/index.php/Arch_User_Repository`

[20] The criteria for this promotion is rather strict. Packages must be stable, have good support for Arch Linux systems, and have no legal concerns. Often times, software copyright and licensing issues are the only factors preventing popular packages from being promoted from the AUR to `Community`.

[21] Trusted Users have a multitude of duties, which are defined in an official bylaws document found at `https://aur.archlinux.org/trusted-user/TUbylaws.html`. There is also a list of guidelines that they must follow in the wiki at `https://wiki.archlinux.org/index.php/AUR_Trusted_User_Guidelines`.

[22] Standard PGP key-retrieval methods involve acquiring the key directly or acquiring the key from a PGP keyserver

activity with this public key will always be considered "valid", since the user has already signed the key. In short, "trust-enforced" mode of operation might entice users to sign keys that they have not actually validated.

In general, the "trust-enforced" mode of the AUR provides a decently strong trust scheme for the AUR. However, it relies heavily on PGP's Web-of-Trust. As pointed out by others[6], the PGP Web-of-Trust does not scale very well to many users. Therefore, it is very difficult to publish a "trust-enforced" package on the AUR and expect all potential users to trust the public-key signature. In fact, a publisher with no PGP-savvy contacts would be unable to deliver his AUR package to a single user, since no user should trust him. In order to circumvent this problem, publishers are allowed to insert a *validpgpkeys* entry into the *PKGBUILD*. The *validpgpkeys* entry is an array of PGP key fingerprints that will automatically be trusted to sign the package. That is, the end-user will go about acquiring the public key in the same way, but `makepkg` will automatically trust any public key with a fingerprint in the *validpgpkeys* array. It is clear that the publisher, who also signed the source-code, will also put his own key fingerprint in the *validpgpkeys* array. Thus, a *PKGBUILD* of this form does not protect against adversarial publishers, but other parties that may have altered the package before it reaches the end-user. I refer to this mode of *PKGBUILD* operation as "trust-provided".[23]

Furthermore, the AUR has the ability to support "trust-enforced" packages; however, it also has to find a balance between trustability and ease-of-use. Therefore, for each AUR package, it is the job of the end-user to determine what trust-mode the *PKGBUILD* operates with and act accordingly. I also propose that `makepkg` be fitted with a mechanism for alerting the user which mode of trust it is running in, with no warning for a validated "trust-enforced", a warning for a validated "trust-provided", and a warning and confirmation for "trustless".

Package Count per Mode of Operation		
Trust-Enforced	**Trust-Provided**	**Trustless**
269	688	37,400

Figure 3: The number of AUR *PKGBUILD* files under each mode of operation.

7.3 Security in the AUR

The essence of using the AUR is compiling and running code that non-trusted users provide. Of course, therefore, the AUR is severely limited in terms of security. It is trivial for a maintainer to upload something malicious to the AUR. Further, it is trivial for a user to download and run something malicious from the AUR. An end-user should always inspect a *PKGBUILD* file to determine what it does and where it retrieves its package from.

The *PKGBUILD* file is a `bash` script. Thus, it could contain any arbitrary code that would not be in the interest of a user. Luckily `makepkg` is not run with elevated privileges. However, a malicious *PKGBUILD* could still inflict serious harm to a user. As an example, running `makepkg` on a malicious *PKGBUILD* file might result in a user's private encryption keys being sent to the maintainer of the package (Let's hope they're encrypted!).[24]

Since installing software is complex, a *PKGBUILD* file has support for install-scripts, which contain functions that are executed at certain points of the install process. This process is ushered by `pacman` and will be executed after `makepkg` has completed. Since the installation process regularly installs system-level software, it must be run as root. The installation scripts specified in the *PKGBUILD* file, therefore, are also run as root. The maintainer of the *PKGBUILD* can thus run arbitrary code with elevated privileges.[25]

Of course, as previously described, the AUR is an inherently insecure place. Therefore, it is expected that the user manually audit the *PKGBUILD* file to determine whether it does something which seems unsafe. Another problem, however, is that a *PKGBUILD* does not contain binary files, but specifies an alternative location to download the files from. Therefore, an end-user may trust the AUR system, but

[23] Details about how a *PKGBUILD* can be arranged can be found in the ArchWiki at `https://wiki.archlinux.org/index.php/PKGBUILD`

[24] A demonstration of such a *PKGBUILD* file can be found at `http://brandonio21.com/security/archlinux/demos/aur/malicious-pkgbuild/PKGBUILD`

[25] An example of such a layout can be found at `http://brandonio21.com/security/archlinux/demos/aur/malicious-install/PKGBUILD`

it is entirely possible that the filehost is compromised or the connection is susceptible to a man-in-the-middle attack. In that case, a remote attacker might change the contents of the binary package (which will be noticed when the end-user runs the software) or an install script (which will be noticed when the end-user simply installs the software). To overcome this, a maintainer might ensure that package files are downloaded only over HTTPS. *PKGBUILD* files also support source-signing (as mentioned above) and source-hashing - both are sufficient for overcoming these attacks, since the PKGBUILD file is hosted on a trusted `archlinux.org` server.

Package Count per Hashing Algorithm	
No Validation	10,275
fnt256	2
md5	15,186
sha1	1,962
sha224	1
sha256	8,933
sha384	19
sha512	1,976
md51	1
sah256	1
sha265	1

Figure 4: The number of AUR *PKGBUILD* files using hash functions to validate against man-in-the-middle attacks. Note that 17,148 packages are using provably insecure hash functions (MD5 and SHA1). The second, lower, table shows usages of invalid hash functions. Note that some packages with "No validation" may actually be running under a "trust-enforced" or "trust-provided" mode, meaning it may have a form of validation that is not explicit hashing.

7.4 Proposals for Increased AUR Security

Since the AUR is insecure by design (in favor of flexibility), there is not much that can be done to make it more secure. At the least, since it is the user's job to manually verify the contents of each *PKGBUILD* file that they interact with, I propose an extension to `makepkg` that first parses the *PKGBUILD* file and forces the user to manually confirm any relevant entries.[26]

The AUR is also fairly weak against active man-in-the-middle attacks. There are no requirements in a *PKGBUILD* file that prevent man-in-the-middle attacks. I propose another extension to `makepkg` which rejects a *PKGBUILD* if it does not contain proper validation methods. That is, if the *PKGBUILD* does not include an HTTPS source address, a trustful (non "trustless") signature scheme, or a secure-hash checksum, it should be rejected.

In addition, the AUR lacks a concept of maintainer accountability. Although users are expected to self-check *PKGBUILD* files, there is no way for them to report a fishy AUR package. At most, the user may leave a comment or mark the package as "out-of-date". However, a package has no way of being reported as being malicious. Further, a package maintainer has no way of being reported. It might be claimed that it is the job of "Trusted Users" to moderate and purge the AUR of any malicious packages, but I propose that users should be able to report a malicious AUR maintainer, whereupon a "Trusted User" might further review the case. Of course, I'm sure some sort of e-mail or IRC communication would accomplish the same thing, but I propose something more user-accessible.

8 Security in the Community

Arch Linux's security model is powered by PGP and the assumption that the user might validate certain steps of the operating system process. A community that is security minded, therefore, should expect to explain certain steps to non-security-minded users. I will now analyze the availability of security information in the Arch Linux community as well as the attitudes of email security announcements.

[26] This security feature is already implemented in `pacaur`, a tool to download from the AUR. However, it only asks the user if they would like to manually inspect install scripts and the *PKGBUILD* file. I claim it would be more useful to present the user with the relevant entries in their entirety

8.1 Availability of Security Information

The primary source of information regarding Arch Linux is the Arch Wiki. Not only does the Arch Wiki contain information about specific programs, but also step-by-step guides on how to perform certain functions, like system installation and administration. In these cases, it is very important that the information contains the necessary security advice.

The ArchWiki page for "Security"[27] contains a good amount of information regarding good system security practices and is updated regularly. It also contains references to other important computer security topics, such as application sandboxing and password management.

There is also lots of information regarding package trust on the "Package Signing" page, which urges users to enforce package signatures.[28] An interesting point, however, is that the Arch Wiki pages are missing information regarding verifying master keys upon installing the system, using only trusted mirrors, and verifying output of `pacman` operation.[29]

8.2 Security Announcements

When a package vulnerability is discovered, Arch Linux is very quick to push an updated version through their package management system. Once the updated version is available, an announcement will be sent to the `arch-security` mailing list, which contains details about the vulnerability and how to upgrade the package.[30] The worry, of course, would be that a malicious mirror would prevent the user from updating their package. However, the security announcement email usually contains instructions on how to update the package using `pacman`. These instructions always contain a "version-guard", something in the form of `pacman -Syu "package_name>=1.2-3"`, which will ensure that the package `package_name` is upgraded. These security announcements, therefore, give instructions which would prevent malicious-mirror database-pinning and package-downgrading.

Unfortunately, however, the only place that these security announcements are delivered is the `arch-security` mailing list. Thus, in order to receive the benefit of knowing which packages to upgrade and when, a user would have to subscribe to this mailing list. This is a manual process, and the number of subscribers to the mailing list will always remain small when compared to the number of Arch Linux users as a whole.

In addition to e-mail announcements, Arch Linux also maintains a list of current known packages with vulnerabilities that is viewable by all (in addition to tools to query the list).[31]

9 Conclusion and Future Work

My primary goal in writing this paper was to determine whether Arch Linux takes security seriously. The design of their trust model, the preventative measures inserted into `pacman`, as well as the wealth of information regarding security on official channels all show that Arch Linux does indeed take security seriously.

Unfortunately, as this research has revealed, there are some spots where the Arch Linux has fallen short. However, my time spent on the forums and mailing lists during this research has also shown that the Arch Linux community is very open to answering questions and fixing flaws. I hope that this research results in security improvements in the Arch Linux ecosystem, especially in Arch Linux's package management system.

As a direct result of this paper, I have begun work on `mspacman`[32], a wrapper around `pacman` that attempts to implement proposals suggested in this paper.

I also plan on bringing issues discovered in this paper to light in the various Arch Linux security mailing lists.

In the larger scope of things, Brown et. al[1] have begun work on designing and implementing a secure package management system that would resolve the problems present in this paper as well as those presented by Cappos et. al.[3]

[27] The ArchWiki page for security can be found at `https://wiki.archlinux.org/index.php/Security`

[28] The ArchWiki page for package signing can be found at `https://wiki.archlinux.org/index.php/Pacman/Package_signing`

[29] The installation guide found at `https://wiki.archlinux.org/index.php/installation_guide#Select_the_mirrors` only mentions the generation of mirrors, but not that they should be trusted.

[30] An archive of this mailing list can be found at `https://lists.archlinux.org/pipermail/arch-security/`

[31] This list is available at `https://security.archlinux.org/`. It would be an interesting exercise to see if this could be used to extend `pacman` into a more secure version.

[32] Progress on `mspacman` can be followed on its GitHub page at `https://github.com/brandonio21/mspacman`

References

[1] Fraser Brown, Ariana Mirian, Atyansh Jaiswal, Andres Ntzli, and Deian Stefan. *SPAM: a Secure Package Manager*. Drafted April 1, 2017.

[2] Nikita Borisov, Ian Goldberg, and Eric Brewer. *Off-the-Record Communication, or, Why Not To Use PGP. WPES'04*. October 28, 2004.

[3] Justin Cappos, Justin Samuel, Scott Baker, and John H Hartman. *A Look In the Mirror: Attacks on Package Managers*. CCS08, October 2731, 2008, Alexandria, Virginia, USA.

[4] Andrey Falko. *Package Manager: The Core of a GNU/Linux Distribution*. Simon's Rock College. 2007.

[5] Allan McRae. *Two PGP Keyrings for Package Management in Arch Linux*. `http://allanmcrae.com/2015/01/two-pgp-keyrings-for-package-management-in-arch-linux/`, 2015. Accessed 31 May 2017.

[6] Mike Perry. *[tor-talk] Why the Web of Trust Sucks*. `https://lists.torproject.org/pipermail/tor-talk/2013-September/030235.html` 29 September 2013. Accessed 31 May 2017.