Conex - establishing trust into repositories

Hannes Mehnert *
University of Cambridge

Louis Gesbert
OCamlPro

Abstract

Software update systems, such as opam, should verify that packages and metadata thereof are authenticated. Opam does not include any mechanism to verify authenticity at the moment. We propose *conex*, which uses end-to-end digital signatures from the package author to the user. Conex is implemented in OCaml and can be integrated into opam to build a security story where even the central repository server does not need to be trusted. Authors and users will not need to change their workflow, whereas janitors (repository maintainers) will have to adjust their workflow.

1. Introduction

Software update systems, such as opam, need to discover, download, and install packages. Crucial to all steps is that the received data is authenticated before being used to prevent attacks injecting backdoors. A software update system relies on a central server which provides a repository (source code and metadata such as dependencies and build recipe) of all packages. When a new package is released, the repository is updated. The central server is an online services and a single point of failures, and recently these servers been attacked [1–6, 8–21, 24–26, 28] to distribute malware.

While so far there is no known breach of opam installations (but other package managers have been compromised, see [6, 13, 15, 25]), there is not much security in place: In opam 1.2.2 none of the steps (discover, download, install) includes verification. The discover and download steps use https, but pass `--no-check-certificates` to the download utility `wget`, which disables certificate verification. Any man-in-the-middle between client and repository can arbitrarily modify updates or tarballs. The build and install steps may do any modifications to the file system, not limited to the build and target directory of the package. In opam 2.0 (not released yet) the download tool verifies the TLS certificates [22] using the system-wide trustchain. Nevertheless, a breach of the repository server enables an attacker to modify updates and tarballs arbitrarily. The (known weak [23]) MD5 digest of source tarballs is stored in the repository as metadata. Neither the build recipe nor additional files are checksummed.

In this paper we present *conex*, a system which uses end-to-end digital signatures (the original author signs their package and build instructions, the client verifies this signature) to re-establish trust into opam. Using end-to-end signature, the central server hosting the repository does not need to be trusted, neither does the server where the tarballs are downloaded from need to be trusted! Conex and opam integrate well, and conex has been designed with opam in mind, but it can be used without opam to publish and update data in a way that it can be authenticated. The main opam repository is a git repository hosted on GitHub. While conex makes use of GitHub identifiers, it does not depend at all on GitHub, and not even on git. Conex can also be used with other public or private opam repositories, only an initial set of trust anchors has to be distributed in a safe manner (e.g. an opam package in the main opam repository). The main goal of conex is to provide a secure way to distribute packages. Another design goal is to retain the workflow of package authors and opam users. Janitors (repository maintainers) need to adapt to a slightly more complex workflow.

Conex is implemented in OCaml using only few dependencies (it should be installable without opam), and licensed under the 2-clause BSD license (source [7]).

Earlier work on securing software update systems, such as the update framework [31] and diplomat [29] focus on distributing directories securely, but make simultaneous updates of several packages by different authors hard, due to centralised files with authorisation and key information). Our earlier proposal for opam signing [27] adjusted ideas from TUF to the current opam workflow, but introduced superfluous structure. Conex still builds on top of tuf and diplomat, but simplifies our original proposal by introducing less structure and being less complex.

In section 2 we will discuss the design of conex, followed by reasonable shortcuts in section 3. Afterwards in section 4 we present the implementation and opam integration. We will discuss related in section 5 and conclude with future work in section 6.

2. Design

We first take a step back and do not consider all the details of opam for now, but focus on how to distribute a repository in a secure way.

Conex does not claim to protect against undiscovered vulnerabilities in the packages, dependent tools, or conex itself. It is also not a scanner for installed vulnerable packages.

2.1 Threat model

There are many risks that users of software update systems face, ranging from interjecting traffic over weaknesses in TLS, weaknesses in the network infrastructure, compromising signing keys, stealing keys, to compromising the repository or signing infrastructure.

This leads us to consider a threat model where parts of the involved systems are compromised. We assume that an attacker can:

- Compromise the central server.
- Respond to user requests (acting as man-in-the-middle).

An attack is successful if the attacker can change the contents of a package that a user installs, or preventing a user from updating to the most recent version of a package.

We include signatures and keys in the same repository as the package metadata, this prevents attackers to hinder a user to receive key updates while updating metadata (possibly signed with a just compromised and revoked key). We embed a monotonic increasing counter in each piece of data to prevent rollback attacks.

* Funded by EPSRC Programme Grant EP/K008528/1

Compromise of (offline!) keys cannot be detected automatically, but requires janitors, authors, and other people to carefully read updates. We assume that authors and janitors protect their keys reasonably. If a key is compromised, we consider conex security to be effective if the impact is limited to the authorised packages of the compromised key.

Both a snapshot of the repository can be verified using a set of initial trust anchors (public keys of janitors), and an update (patch) to an existing verified repository which leads to a verified repository with the patch applied.

2.2 Roles

There are two roles in conex:

- An *author* who develops software and releases it into the repository.
- A *janitor* who keeps the repository tidy and in a working state (fixing up reverse dependencies etc.). Janitors also maintain the package name and author name resources.

An author signs their own packages, and janitors have to introduce authors and authorise authors for their packages. A janitor has more privileges, and is also likely be an author of their packages. In section 3 we discuss some ways to make conex easier to use, leveraging the security level.

If a single janitor would be able to adjust packages which are missing version constraints (due to new releases of dependencies), revoke or update an author's key, and/or modify the set of janitors, then compromising a single janitor key would be sufficient to maliciously update the repository. We avoid this by requiring a quorum of janitors to sign updates.

2.3 Resources

The repository contains several resources: package metadata (checksums, build recipes, dependencies), author (public key, id, role) and authorisation (relation between package name and author). The role field distinguishes janitors and authors.

Each resource is only valid if it is signed by either one of the authorised key identifiers, or a quorum of janitors. Each resource contains a monotonic counter to prevent rollback attacks.

Packages, identified by their name, and key identifiers, are the names we need to protect against starvation attacks. Janitors need to faithfully verify new package and public key submissions.

Resource updates are distributed (each author updates their own packages), and should be free of conflicts (conflicts are detected during merge, it is up to the janitors to decide which update should go through, and which should be resubmitted on top of the other).

2.4 Repository layout

While conex is not bound to a file system, we'll reuse their terminology. The root directory has two subdirectories, `keys` and `data`.

The `keys` directory contains one file, named after the key identifier. The `data` directory consists of subdirectories, which each must contain a file `authorisation` and a file `index`, plus possibly any subdirectories. Each subdirectory must be listed in the `index` file, and contains a `checksum` file besides any other data files and directories (all listed in the checksum file).

Each file contains a set of key-value pairs. Any key-value representation which can be non-ambiguously normalised into a string, can be used (such as the opam format, json, s-expressions). Each file contains a `signed` structure: `(signed:` **resource**`, sigs:` **(id, signature) list**`)`. Resources, either a public key, an authorisation, or a checksum, have been mentioned before, and are explained in more detail below. The signature is computed over the normalised string representation of the resource concatenated with the public key id (separated by a space character).

id: 7bit ASCII case sensitive string
name: 7bit ASCII case sensitive string
signature: Base64 encoded RSASSA-PSS with SHA256 (PKCS1)
public key: RSA public key ($\geq$ 2048 bit, PKCS1, PEM encoded)
role: `Janitor` | `Author` | `Other` of **id**
service: `EMail` of **name** | `GitHub` of **name** | `Other` of **id** * **name**
digest: SHA256 digest of file
byte size: size as **Int64** of file

Figure 1. Data types for conex version 0

counter: **Int64**,
version: **Int64**,
key: **public key**,
identifier: **id**,
accounts: **service list**,
role: **role**

Figure 2. Public key

counter: **Int64**,
version: **Int64**,
name: **name**,
authorised: **id list**

Figure 3. Authorisation

counter: **Int64**,
version: **Int64**,
name: **name**,
releases: **name list**

Figure 4. Index

counter: **Int64**,
version: **Int64**,
name: **name**,
files: **(name, byte size, digest) list**

Figure 5. Checksum

The data types for version 0 of conex are presented in Figure 1. Distinction between id and name is only for clarity (ids are the public key ids, where names are the package names), they form two disjoint sets of identifiers. Upon insertion of new identifiers, checks for non-collision and validity need to be done. The file formats of keys (Figure 2), authorisations (Figure 3), index (Figure 4), and checksums (Figure 5) include a version number (0 for this proposal) and a monotonic counter (starting at 0) each. The index is signed by an authorised author, and contains a list of releases of the given package name. The list of files in the checksum contains a list of all (recursively) present files in the current directory. Files not mentioned in the checksum are not exposed from conex to opam. Instead of removing author ids and packages, we put empty placeholders signed by janitors into the repository. Otherwise an attacker can remove all the author ids and packages, since there's no signed trace of this change.

2.5 Chain of Trust

A package is valid if and only if the chain of trust can be verified. All files in the subdirectory are registered in the checksum file, which is signed by either an author (listed in the authorisation file in the parent directory) or a quorum of janitors. The subdirectory name must be present in the valid index file of the parent directory. The authorisation file itself has to be signed by a quorum of janitors (whose keys need to be in the repository and appropriately signed). Public keys have to be signed by a quorum of janitors as well.

To bootstrap the chain, we assume an existing set of janitors (which cardinality is above the quorum). The set for the main opam repository is distributed with opam, thus a compromise of opam is fatal (out of scope for conex, need to be authenticated via other means).

2.6 Snapshot verification

Verification of a snapshot is straightforward: take the set of initial janitors as trusted, then iterate through all packages, verify the authorisation file (properly signed by a quorum of janitors), and verify all checksum files. The public keys are verified upon demand

(whenever a signature is encountered, the key of the signing id is verified).

2.7 Update verification

Consider that the repository in the old state is verified, use it to verify the update (in form of a patch) in the following way:

- key modification – key is sufficiently big, unique id matching filename, counter is increased, signed by quorum, no signature with old key
- key insertion – key is sufficiently big, unique id matching filename, counter is 0, signed by quorum
- key deletion – counter increasing, no signatures with this key, empty string is used to denote no key, signed by quorum
- role modification – counter increasing, no signature with old role, signed by quorum
- authorisation insertion – counter is 0, id matches directory, signed by quorum, authorised key ids are present
- authorisation modification – counter increasing, all checksum are signed with an authorised key, signed by quorum
- complete package removal – empty authorised, empty releases, signed by quorum
- package insertion – valid authorisation, index, and checksum, both counter 0, signed by author (authorisation by quorum)
- package update – valid checksum, counter increasing, signed by author
- new package version release – index counter increasing and new version added to list, valid checksum, signed by author
- package version deletion – counter increasing, removed from releases, signed by author

2.8 Security Analysis

Assuming a man-in-the-middle-attacker between the repository and a user wants to deliver a backdoored package. The first step is to insert a backdoor, which can either be done by modifying the source code of a package (either updating the source tarball, preparing a new release, or introducing a separate patch file) in the repository, or introducing a new package and a dependency. All these mechanism require either the original author key (or a quorum of janitors). Any suspicious behaviour in widely used packages will likely be spotted by the community which reads through the commit logs.

An attacker can freeze a client from updating (by preventing the client to communicate with the repository server), but downgrades of packages are not possible (an update cannot remove packages without appropriate signatures).

Mix and match (providing outdated versions of package A, newer of package B, up-to-date package C) are not mitigated in conex since there is no central list of releases. In the next section, we will discuss a timestamping service which mitigates mix and match attacks (and let a user detect freeze attacks).

Conex does not depend on unreliable system time and comparison of timestamps, instead it uses monotonic counters. It handles revocation explicitly by retaining the resource in the repository, but leaving the data element (e.g. public key) empty. Other systems rely on git signing, which is out-of-tree data and thus an attacker can prevent a user from receiving this data (while still receiving (potentially malicious) updates).

In the case conex cannot verify package metadata, it will not pass this unverified data to opam. In case conex cannot verify a downloaded tarball, this will be reported.

The weakest link of security are humans and their computer systems - janitors and authors will have their keys compromised after serving some website or inserting a USB stick.

3. Shortcuts

In the presented setup, conex requires verification of at least 8 signatures (3 for the public key, 3 for the authorisation, then index and checksum) per package. Valid public key can be cached during processing, leaving the number at 5, which is still too high, especially for embedded systems.

In addition, the previously mentioned mix and match and freeze attack are possible. To mitigate this, a *timestamp service*, which updates the repository regularly, verifies all patches, and attaches its digital signature to the latest patch (via an git annotated tag over the git commit). The public key of the timestamp service is inside the repository and signed by multiple janitors. Clients can, instead of verifying packages and updates independently, trust the timestamp service (or multiple timestamp services). This will reduce computational overhead (signature verification are expensive operations on big numbers), and might be worth for embedded devices. If there are multiple timestamp services run by different organisations, and its code is well audited, it might be reasonable to use them as default. While there is not yet an implementation of a timestamp service, it will likely be implemented as a MirageOS [30] application, thus the trusted code base is small.

The main opam repository served as an example over the last section, but conex can easily be applied to any other opam repository. Initial set of janitors (can be a single element), the quorum (can be one) must be distributed for each other repository. Of course, an automated timestamp service can be used for other repositories as well. Conex can also securely distribute binary repositories (since OCaml 4.03 it emits reproducible binaries).

If the work overhead for janitors turns out too high, we can leverage the system in a way that *neither authorisation nor public keys require a quorum*. This will open the system to name squatting attacks, and in a snapshot authorisations cannot be verified that they contain the same information as the original ones, but updates can be verified (and again, we can outsource trust into the timestamping service).

Another useful automated system will be a *PR bot*, similar to Camelus, which verifies each PR on the main opam repository and reports the results of the update (whether it is valid, how many janitors need to sign, etc.).

4. Conex implementation

An early version of conex [7] is implemented in OCaml, but not yet released at the time of writing. It does not depend on opam-lib, but instead will be functorised over the data layout (json, opam, s-expression). To further minimise the dependencies (installed via opam), the goal is to have a cryptographic module which uses the widely deployed openssl utility for bootstrapping.

The timestamp service and other automated tools will use the conex library as a basis.

Conex is not yet integrated into opam at all (but some work on opam 2.0 will make integration straightforward). The idea is to use conex as a separate application which proxies discover and download commands done by opam.

Janitors need more tooling support, especially to sign off quorums. The idea is to have a git branch per update requiring a quorum, and the janitors adding their signatures to that branch as separate commits. Once the quorum is reached, the branch can be squashed and merged into the repository. A queue of updates needing attention (a quorum) will be provided by the tooling.

5. Related work

Conex is based on the update framework [31], which centralises all packages into a single `target.txt` file, and has more structure in keys. Instead of doing this, conex was designed with version control and distributed updates in mind.

A more recent contribution is Diplomat [29], but instead of automatically promoting unclaimed packages to claimed (which we can do using timestamp services), we believe in end-to-end signing of packages. Our security model relies on offline keys which are protected by authors.

The Haskell signing proposal and implementation does not enforce authors to sign their packages, but instead the repository is signed. Thus, the repository server (or key server) needs to be trusted ultimately.

6. Conclusion

We presented conex: securing opam, a software update system, without modifying the current workflows too much. The security will be based on offline keys and end-to-end signing of packages. This removes any central server from the trust chain. Attacks, observed in practise, and theoretic from related work, are mitigated by conex.

Future work includes finishing conex, writing more tests, integration into opam, and develop automated services.

References

[1] Adobe to revoke code signing certificate https://blogs.adobe.com/conversations/2012/09/adobe-to-revoke-code-signing-certificate.html.

[2] apache.org incident report for 04/09/2010 https://blogs.apache.org/infra/entry/apache_org_04_09_2010.

[3] apache.org incident report for 08/28/2009 https://blogs.apache.org/infra/entry/apache_org_downtime_report.

[4] Attackers sign malware using crypto certificate stolen from opera software http://arstechnica.com/security/2013/06/attackers-sign-malware-using-crypto-certificate-stolen-from-opera-software/.

[5] An attempt to backdoor the kernel http://lwn.net/Articles/57135/.

[6] Cabal: Packages are downloaded insecurely https://github.com/haskell/cabal/issues/936.

[7] Conex git repository https://github.com/hannesm/conex.

[8] The cracking of kernel.org http://www.linuxfoundation.org/news-media/blogs/browse/2011/08/cracking-kernelorg.

[9] Debian investigation report after server compromises https://www.debian.org/News/2003/20031202.

[10] Freebsd.org intrusion announced 17-11-2012 http://www.freebsd.org/news/2012-compromise.html.

[11] A further update on php.net http://php.net/archive/2013.php#id2013-10-24-2.

[12] Gnu savannah compromise https://savannah.gnu.org/maintenance/Compromise2010/.

[13] How to take over the computer of any java (or clojure or scala) developer http://blog.ontoillogical.com/blog/2014/07/28/how-to-take-over-any-java-developer/.

[14] Important: Information regarding savannah restoration for all users https://savannah.gnu.org/forum/forum.php?forumid=2752.

[15] Newly paranoid maintainers http://blog.npmjs.org/post/80277229932/newly-paranoid-maintainers.

[16] The php project and code review http://bjori.blogspot.com/2010/12/php-project-and-code-review.html.

[17] phpmyadmin corrupted copy on korean mirror server https://sourceforge.net/blog/phpmyadmin-back-door/.

[18] php.net security notice http://www.php.net/archive/2011.php#id2011-03-19-1.

[19] Public key security vulnerability and mitigation https://github.com/blog/1068-public-key-security-vulnerability-and-mitigation.

[20] Recent reboots explained: the forge has been compromised https://forge.ocamlcore.org/forum/forum.php?forum_id=913.

[21] Red hat security advisory: openssh security update https://rhn.redhat.com/errata/RHSA-2008-0855.html.

[22] remove insecure / no-check-certificate flags https://github.com/ocaml/opam/pull/2460.

[23] Rfc 6151: Updated security considerations for the md5 message-digest and the hmac-md5 algorithms https://tools.ietf.org/html/rfc6151.

[24] rsync.gentoo.org: rotation server compromised https://security.gentoo.org/glsa/200312-01.

[25] Rubygems data verification http://blog.rubygems.org/2013/01/31/data-verification.html.

[26] Security incident on fedora infrastructure on 23-01-2011 https://lists.fedoraproject.org/pipermail/announce/2011-January/002911.html.

[27] Signing the opam repository https://opam.ocaml.org/blog/Signing-the-opam-repository/.

[28] Wordpress passwords reset https://wordpress.org/news/2011/06/passwords-reset/.

[29] KUPPUSAMY, T. K., TORRES-ARIAS, S., DIAZ, V., AND CAPPOS, J. Diplomat: Using delegations to protect community repositories. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, Mar. 2016), USENIX Association, pp. 567–581.

[30] MADHAVAPEDDY, A., MORTIER, R., ROTSOS, C., SCOTT, D., SINGH, B., GAZAGNAIRE, T., SMITH, S., HAND, S., AND CROWCROFT, J. Unikernels: Library operating systems for the cloud. In *ASPLOS* (2013), ACM.

[31] SAMUEL, J., MATHEWSON, N., CAPPOS, J., AND DINGLEDINE, R. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 61–72.