Secure Sharing Between Untrusted Users in a
Transparent Source/Binary Deployment Model

STC / ASE 2005

Eelco Dolstra
eelco@cs.uu.nl

Universiteit Utrecht, Faculty of Science,
Department of Information and Computing Sciences

October 20, 2005
Create a *package management system* that allows *any user* to install software.
Package management models

### Traditional Unix package managers

- RPM, Apt, FreeBSD Ports, Gentoo Portage, ...
- Manage dependencies
- Only the administrator can install packages
- ... since they go into global directories like `/usr/bin`
- Packages are *shared* between users

### Monolithic packaging systems

- Windows, Mac OS X
- Everybody can install packages
- But there is no sharing (unless explicitly arranged)
<table>
<thead>
<tr>
<th>Traditional Unix package managers</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ RPM, Apt, FreeBSD Ports, Gentoo Portage, ...</td>
</tr>
<tr>
<td>▶ Manage dependencies</td>
</tr>
<tr>
<td>▶ Only the administrator can install packages</td>
</tr>
<tr>
<td>▶ ... since they go into global directories like <code>/usr/bin</code></td>
</tr>
<tr>
<td>▶ Packages are <em>shared</em> between users</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monolithic packaging systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Windows, Mac OS X</td>
</tr>
<tr>
<td>▶ Everybody can install packages</td>
</tr>
<tr>
<td>▶ But there is no sharing (unless explicitly arranged)</td>
</tr>
</tbody>
</table>
Why do we want sharing?

- More efficient use of resources
- Especially due to common dependencies: $\Theta(N + M)$ instead of $\Theta(N \times M)$

The problem

- Users may be mutually untrusted
- If Alice installs Firefox, then Bob may not want to use it; it may contain a Trojan horse

Typical untrusted environments

- Student login servers
- Hosting providers
- Computational grids
Why do we want sharing?

- More efficient use of resources
- Especially due to common dependencies: \( \Theta(N + M) \) instead of \( \Theta(N \times M) \)

The problem

- Users may be mutually *untrusted*
- If Alice installs Firefox, then Bob may not want to use it; it may contain a Trojan horse

Typical untrusted environments

- Student login servers
- Hosting providers
- Computational grids
Why do we want sharing?

- More efficient use of resources
- Especially due to common dependencies: $\Theta(N + M)$ instead of $\Theta(N \times M)$

The problem

- Users may be mutually *untrusted*
- If Alice installs Firefox, then Bob may not want to use it; it may contain a Trojan horse

Typical untrusted environments

- Student login servers
- Hosting providers
- Computational grids
This paper extends the *Nix deployment system* to support secure sharing between untrusted users.
Central idea: store all components in isolation.

Unique paths:

/nix/store/jjp9pirx8b3nqs9k...-firefox

which is an SHA-256 hash of **all** inputs used to build the component:

- Sources
- Libraries
- Compilers
- Build scripts
- Build parameters
- System type
- ...

- **Prevent** undeclared **build time** dependencies.
- **Scan** for **runtime** dependencies.
- Deploy only **closures** under the **depends-on** relation.
Nix store

Unique paths for different versions

- bd6593219f8dcb63...-gtk+-2.2.4
  - lib
    - libgtk-x11-2.0.so.0
- ce2d7d2a41456bab...-wxGTK-2.4.2
  - lib
    - libwx_gtk2-2.4.so
- e889db0595672287...-wxPython-2.4.2.4
  - (lots of Python bindings)
- 9ed8c4231bfde4af...-bittorrent-3.4.2
  - bin
    - btdownloadgui.py
- 300ccc1a41af3abc...-gtk+-2.4.13
  - lib
    - libgtk-x11-2.0.so.0
- f51ec7d5663c735e-zapping-0.7.3
  - bin
    - zapping
Nix expressions

```nix
derivation {
  name = "firefox-1.0.7";
  builder = ./builder.sh;
  src = fetchurl {
    url = http://.../firefox-1.0.7-source.tar.bz2;
    md5 = "5704a8c36de84b408e069afb0c5bc1df";
  };
  pkgconfig = derivation { ... };
  gtk = derivation { ... };
}
```
firefox.nix

derivation {
    name = "firefox-1.0.7";
    builder = ./builder.sh;
    src = fetchurl {
        url = http://.../firefox-1.0.7-source.tar.bz2;
        md5 = "5704a8c36de84b408e069af0c5bc1df";
    };
    pkgconfig = derivation { ... };
    gtk = derivation { ... };
}
source $stdenv/setup

PATH=$pkgconfig/bin:$PATH

tar xvfj $src
cd firefox-*
./configure --prefix=$out --with-gtk=$gtk
make
make install
source $stdenv/setup

PATH=$pkgconfig/bin:$PATH

tar xvfj $src
cd firefox-*
./configure --prefix=$out --with-gtk=$gtk
make
make install

Environment variables pass locations of dependencies, e.g. /nix/store/0z017z...-pkgconfig
source $stdenv/setup

PATH=$pkgconfig/bin:$PATH

tar xvfj $src
cd firefox-*
./configure --prefix=$out --with-gtk=$gtk
make
make install

Holds the component’s path in the Nix store, e.g. /nix/store/jjp9pi...-firefox
To build and install Firefox:

$ nix-env -f firefox.nix -i firefox

The path of Firefox (e.g., /nix/store/jjp9pi...-firefox) is added to the user’s PATH environment variable.
To build and install Firefox:

```bash
$ nix-env -f firefox.nix -i firefox
```

The path of Firefox (e.g., `/nix/store/jjp9pi...-firefox`) is added to the user’s `PATH` environment variable.
Nix expressions give a source deployment model.
We get binary deployment by sharing pre-built components.

On the producer side:

```
$ nix-push $(nix-instantiate firefox.nix) \ 
    http://server/cache
```

On the client side:

```
$ nix-pull http://server/cache
$ nix-env -f firefox.nix -i firefox
```

```
nix-pull registers substitutes:
“if I need to build path /nix/store/jjp9pi...-firefox, I can download and unpack http://example.org/jjp9pi...-firefox.nar.bz2 instead”
```
Nix expressions give a **source deployment model**.

We get **binary deployment** by sharing pre-built components.

On the producer side:

```
$ nix-push $(nix-instantiate firefox.nix) \ http://server/cache
```

On the client side:

```
$ nix-pull http://server/cache
$ nix-env -f firefox.nix -i firefox
```

*nix-pull registers substitutes:*

“if I need to build path `/nix/store/jjp9pi...-firefox`, I can download and unpack `http://example.org/jjp9pi...-firefox.nar.bz2` instead”
Nix expressions give a **source deployment model**.

We get **binary deployment** by sharing pre-built components.

On the producer side:

```bash
$ nix-push $(nix-instantiate firefox.nix) \n http://server/cache
```

On the client side:

```bash
$ nix-pull http://server/cache
$ nix-env -f firefox.nix -i firefox
```

`nix-pull` registers **substitutes**:

“if I need to build path `/nix/store/jjp9pi...-firefox`, I can download and unpack `http://example.org/jjp9pi...-firefox.nar.bz2` instead”
Nix expressions give a **source deployment model**.

We get **binary deployment** by sharing pre-built components.

On the producer side:

```
$ nix-push $(nix-instantiate firefox.nix) \ 
   http://server/cache
```

On the client side:

```
$ nix-pull http://server/cache
$ nix-env -f firefox.nix -i firefox
```

**nix-pull** registers *substitutes*:

“if I need to build path `/nix/store/jjp9pi...-firefox`, I can download and unpack

`http://example.org/jjp9pi...-firefox.nar.bz2` instead”
Goal

Allow untrusted users to run Nix commands, e.g. installation — with sharing

- Users do not have direct write permission to the store
- Build/installation actions are performed by a system user on behalf of users
  - I.e., nix-env is a setuid program or talks to a daemon
- Intended security property: if a Nix expression is trusted, then so is the binary installed by nix-env -i
## Goal

Allow untrusted users to run Nix commands, e.g. installation — *with sharing*

- Users do not have direct write permission to the store
- Build/installation actions are performed by a *system user* on behalf of users
  - I.e., `nix-env` is a *setuid* program or talks to a daemon
- Intended security property: if a Nix expression is trusted, then so is the binary installed by `nix-env -i`
### Sharing in Nix: Example

<table>
<thead>
<tr>
<th>Alice</th>
<th>Nix store</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Gets <code>firefox.nix</code> from trusted source</td>
<td>/nix/store</td>
</tr>
</tbody>
</table>
| - Runs `nix-env -i firefox`  
  Computes path: `/nix/store/jjp9pi...-firefox`  
  Builds it |   |
|       |   |
### Alice

- Gets `firefox.nix` from trusted source
- Runs `nix-env -i firefox`
  
  Computes path: `/nix/store/jjp9pi...-firefox`
  
  Builds it

### Nix store

```
/nix/store
  └── jjp9pi...-firefox
      ├── bin
      │   └── firefox
      └── lib
          └── libxpcom.so
              └── libmozz.so
                  └── ...
```
### Sharing in Nix: Example

<table>
<thead>
<tr>
<th>Alice</th>
<th>Nix store</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Gets <strong>firefox.nix</strong> from trusted source</td>
<td></td>
</tr>
<tr>
<td>- Runs <code>nix-env -i firefox</code></td>
<td><code>/nix/store/jjp9pi...-firefox</code></td>
</tr>
<tr>
<td></td>
<td>- <strong>bin</strong></td>
</tr>
<tr>
<td></td>
<td>- <em>firefox</em></td>
</tr>
<tr>
<td></td>
<td>- <strong>lib</strong></td>
</tr>
<tr>
<td></td>
<td>- <em>libxpcom.so</em></td>
</tr>
<tr>
<td></td>
<td>- <em>libmozz.so</em></td>
</tr>
<tr>
<td></td>
<td>- ...</td>
</tr>
<tr>
<td></td>
<td>Builds it</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>- Gets <strong>firefox.nix</strong> from trusted source</td>
<td></td>
</tr>
<tr>
<td>- Runs <code>nix-env -i firefox</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Already present!</td>
</tr>
</tbody>
</table>
### Sharing in Nix: Example

<table>
<thead>
<tr>
<th>Alice</th>
<th>Nix store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets <code>firefox.nix</code> from trusted source</td>
<td><code>/nix/store/ jjp9pi...-firefox</code></td>
</tr>
<tr>
<td>Runs <code>nix-env -i firefox</code></td>
<td><code>bin</code></td>
</tr>
<tr>
<td>Computes path:</td>
<td><code>firefox</code></td>
</tr>
<tr>
<td><code>/nix/store/ jjp9pi...-firefox</code></td>
<td><code>lib</code></td>
</tr>
<tr>
<td>Builds it</td>
<td><code>libxpcos.so</code></td>
</tr>
<tr>
<td></td>
<td><code>libmozz.so</code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td>Already present!</td>
</tr>
</tbody>
</table>

**Bob**

- Gets `firefox.nix` from trusted source
- Runs `nix-env -i firefox`  
  Computes path:  
  `/nix/store/ jjp9pi...-firefox`  
  *Already present!*
Sharing in Nix: Example

Alice
- Gets `firefox.nix` from trusted source
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/jjp9pi...-firefox`
  Builds it

Bob
- Gets `firefox.nix` from trusted source
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/jjp9pi...-firefox`
  *Already present!*

Carol
- Gets a *different* `firefox.nix`
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/x64bxa...-firefox`
  Builds it

Nix store
```
/nix/store
  jjp9pi...-firefox
    bin
    firefox
    lib
    libxpcom.so
    libmozz.so
    ...
```
Sharing in Nix: Example

Alice
- Gets `firefox.nix` from trusted source
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/jjp9pi...-firefox`
  Builds it

Bob
- Gets `firefox.nix` from trusted source
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/jjp9pi...-firefox`
  *Already present!*

Carol
- Gets a *different* `firefox.nix`
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/x64b xp...-firefox`
  Builds it
Attack method: interfere with local builds

Alice

- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  Computes path:
  `/nix/store/jjp9pi...-firefox`
  Builds it

Nix store

`/nix/store/...`
Attack method: interfere with local builds

Alice

- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/jjp9pi...-firefox`
  Builds it

Nix store

```
/nix/store
    jjp9pi...-firefox
        bin
        lib
            libxpcom.so
            libmozz.so
            ...
```
**Attack method: interfere with local builds**

**Bob**
- Writes `evil.nix`
- Runs `nix-env -i evil`
  - Computes path: `/nix/store/01qr9w...-evil`

**Alice**
- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  - Computes path: `/nix/store/jjp9pi...-firefox`
  - Builds it

**Nix store**
```
/nix/store
  └── j jp9pi...-firefox
      └── bin
          └── firefox
      └── lib
          └── libxpcom.so
          └── libmozz.so
          ...  
```
Attack method: interfere with local builds

Alice
- Gets firefox.nix
- Runs nix-env -i firefox
  Computes path: /nix/store/jjp9pi...-firefox
  Builds it

Bob
- Writes evil.nix
- Runs nix-env -i evil
  Computes path: /nix/store/01qr9w...-evil

Builder of evil.nix
```sh
#! /bin/sh
cp trojan-horse
/nix/store/jjp9pi...-firefox/bin/firefox
```

Nix store
```
/nix/store
  jjp9pi...-firefox
    bin
    firefox
    lib
      libxpcom.so
      libmozz.so
      ...
```
Attack method: interfere with local builds

**Alice**
- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  - Computes path: `/nix/store/jjp9pi...-firefox`
  - Builds it

**Bob**
- Writes `evil.nix`
- Runs `nix-env -i evil`
  - Computes path: `/nix/store/01qr9w...-evil`
  - Builds it

**Nix store**
- `/nix/store`
  - `jjp9pi...-firefox`
    - `bin`
    - `firefox`
    - `lib`
      - `libxpcom.so`
      - `libmozz.so`
      - ...
  - `01qr9w...-evil`
    - ...

Attack method: interfere with local builds

**Alice**
- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  Computes path: `/nix/store/jjp9pi...-firefox`
  Builds it

**Bob**
- Writes `evil.nix`
- Runs `nix-env -i evil`
  Computes path: `/nix/store/01qr9w...-evil`
  Builds it

**Nix store**
```
/nix/store
  jjp9pi...-firefox
    bin
    lib
    ... 
  nmozz.so
  ... 
  01qr9w...-evil
    ... 
```

Skull and crossbones icon
Solution

Isolate builders

- Run each build under a unique user ID (uid)
- I.e., maintain a pool of build users:
  nix-build-1, nix-build-2, ...
- No two uids are used simultaneously
  - Kill all processes running under a uid before using that uid
Alice

▶ Gets firefox.nix
▶ Pulls from evil.org
▶ Runs nix-env -i firefox

Computes path:
/nix/store/jjp9pi...-firefox
Fake substitute is downloaded

Nix store

/nix/store

...
Attack method: register fake substitutes

Alice

- Gets **firefox.nix**
- Pulls from **evil.org**
- Runs `nix-env -i firefox`
  Computes path:
  `/nix/store/jjp9pi...-firefox`
  Fake substitute is downloaded

**http://evil.org/**
Contains Trojan horse substitute
**jjp9pi...-firefox.nar.bz2.**

Nix store
/nix/store
...
Attack method: register fake substitutes

Alice
- Gets firefox.nix
- Pulls from evil.org
- Runs nix-env -i firefox
  Computes path:
  /nix/store/jjp9pi...-firefox
  Fake substitute is downloaded

http://evil.org/
Contains Trojan horse substitute jkp9pi...-firefox.nar.bz2.

Nix store
/nix/store
  jkp9pi...-firefox
    bin
    lib
    libxpcom.so
    libmozz.so
    ...

Fake substitute is downloaded
Attack method: register fake substitutes

**Bob**
- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  
  Computes path: `/nix/store/jjp9pi...-firefox`
  
  Already present!

- Runs Firefox — 0wned!

**Alice**
- Gets `firefox.nix`
- Pulls from `evil.org`
- Runs `nix-env -i firefox`
  
  Computes path: `/nix/store/jjp9pi...-firefox`
  
  Fake substitute is downloaded

**http://evil.org/**
Contains Trojan horse substitute `jjp9pi...-firefox.nar.bz2`.

**Nix store**
```
/nix/store
  jjp9pi...-firefox
  
  bin
  
  lib
  
  libxpcom.so
  
  libmozz.so
  ...
```

Nix store contains a Trojan horse substitute downloaded by Bob.
Attack method: register fake substitutes

**Alice**
- Gets `firefox.nix`
- Pulls from `evil.org`
- Runs `nix-env -i firefox`
  
  Computes path: `/nix/store/jjp9pi...-firefox`
  
  Fake substitute is downloaded

**Bob**
- Gets `firefox.nix`
- Runs `nix-env -i firefox`
  
  Computes path: `/nix/store/jjp9pi...-firefox`
  
  *Already present!*
- Runs Firefox — *0wned!*

**http://evil.org/**
Contains Trojan horse substitute `jjp9pi...-firefox.nar.bz2`.

**Nix store**
- `/nix/store`
  - `jjp9pi...-firefox`
    - `bin`
    - `lib`
      - `libxpcom.so`
      - `libmozz.so`
      - `...`
Attack method: register fake substitutes

Alice
- Gets firefox.nix
- Pulls from evil.org
- Runs nix-env -i firefox
  Computes path: /nix/store/jjp9pi...-firefox
  Fake substitute is downloaded

Bob
- Gets firefox.nix
- Runs nix-env -i firefox
  Computes path: /nix/store/jjp9pi...-firefox
  Already present!
- Runs Firefox — 0wned!

Nix store
- /nix/store/jjp9pi...-firefox
- bin
- lib
  - libxpcom.so
  - libmozz.so
  ...

http://evil.org/
- Contains Trojan horse substitute jkp9pi...-firefox.nar.bz2.
The problem

- We must trust that the substitute (binary) corresponds to the derivation (source) it claims to have been built from.
- The output path of a derivation (like /nix/store/jjp9pi...-firefox) is computed in advance.
- There can be only one /nix/store/jjp9pi...-firefox in the file system at any given time.
  - Extensional model: all contents are assumed to be interchangeable.
  - ... but they are not due to malicious substitutes.
- Thus the trust relation must be established globally, for all users.
Solution: A content-addressable Nix store

- **Content-addressability**: the contents of an component in the store determine its file name

- **Example:**
  - If the contents of a component have hash `j153hbg6n21c...`
  - Then it will be stored in `/nix/store/j153hbg6n21c...`

- **Result**: if two components are equal, they are stored only once

- **Intensional model**: the hash in a path relates to the extensional behaviour of a component

- **This model makes no assumptions that might not hold**: content-addressability is a verifiable security invariant
Building in the content-addressable Nix store

Problem

Component store paths are no longer known in advance. But we need an output path!

Solution

- Use a temporary path with a random hash component, e.g.
  \[
  \texttt{\$out} = /nix/store/0f9hrdwh3nd3...-firefox
  \]
- Run the builder
- Compute the hash $H$ over the output, e.g
  \[
  H = j153hbg6n21c...
  \]
- Rename the temporary path to /nix/store/$H$-name, e.g.
  /nix/store/j153hbg6n21c...-firefox
Self-references

Problem
Components can contain references to their own path.

Example: /nix/store/0f9hrdwh3nd3...-firefox/bin/firefox

```bash
#!/bin/sh
...
moz_libdir=/nix/store/0f9hrdwh3nd3...-firefox/lib/...
...
dist_bin="$moz_libdir"
...
"$dist_bin/run-mozilla.sh" $script_args
"$dist_bin/$MOZILLA_BIN" "$@
```
Self-references (cont’d)

[nix/store/0f9hrdwh3nd3...-firefox/bin/firefox]

... 0a 6d 6f 7a 5f 6c 69 62 64 3d 2f 6e 69 78 |.moz_libdir=/nix|
2f 73 74 6f 72 65 6c 69 62 2f 66 69 72 65 6f 78 2f |/store/0f9hrdwh3|
6e 64 33 6d 7a 35 6c 69 62 64 3d 2f 6e 69 78 |nd3mz5cqcncly5bw|
39 32 35 68 73 65 6c 69 62 2f 66 69 72 65 6f 78 2f |925yh56-firefox/|
6c 69 62 2f 66 69 72 65 6f 78 2f 6e 69 78 2f |lib/firefox-1.4.|
31 0a 4d 52 4e 54 4f 44 4f 4e 73 74 6f 72 65 2e |d3mz5cqcncly5bw9|
...  

Solution

- Compute hashes modulo self-references:
  when computing the final hash, replace every occurrence of the temporary hash by zeroes
- Rewrite occurrences of the temporary hash to the final hash
- Does this work? Yes!
Self-references (cont’d)

/nix/store/0f9hrdwh3nd3…-firefox/bin/firefox

...  
0a 6d 6f 7a 5f 6c 69 62 64 69 72 3d 2f 6e 69 78  
| .moz_libdir=/nix|  
2f 73 74 6f 72 65 2f 00 00 00 00 00 00 00 00 00  
| /store/00000000|  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
| 0000000000000000|  
00 00 00 00 00 00 00 00 2d 66 69 72 65 66 6f 78 2f  
| 0000000-firefox/|  
6c 69 62 2f 66 69 72 65 66 6f 78 2f 31 2e 34 2e  
| lib/firefox-1.4.|  
31 0a 4d 52 4e 4f 4e 53 49 6f 6e 2d 66 6f 72 65 66 6f 78 2f  
| 1.MRE_HOME=/nix/|  
73 74 6f 72 65 2f 30 66 39 68 72 64 77 68 33 6e  
| store/0f9hrdwh3n|  
64 33 6d 7a 35 63 63 6c 79 35 62 77 39  
| d3mz5cqcncly5bw9|  
...

Solution

▶ Compute hashes *modulo self-references*:  
when computing the final hash, replace every occurrence of the temporary hash by zeroes

▶ *Rewrite* occurrences of the temporary hash to the final hash

▶ Does this work? Yes!
Self-references (cont’d)

Solution

- Compute hashes *modulo self-references*:
  when computing the final hash, replace every occurrence of the temporary hash by zeroes

- *Rewrite* occurences of the temporary hash to the final hash

- Does this work? Yes!
Self-references (cont’d)

/nix/store/0f9hrdwh3nd3...-firefox/bin/firefox

... 0a 6d 6f 7a 5f 6c 69 62 64 69 72 3d 2f 6e 69 78 |
     \.moz_libdir=/nix|
 2f 73 74 6f 72 65 2f 6a 31 35 33 68 6b 67 36 6e |
   /store/j153hbg6n|
32 31 63 62 33 79 6d 79 6b 79 64 70 78 36 6b |
  21cb3ymykbydpx6k|
32 63 39 64 78 70 34 2d 66 69 72 65 66 6f 78 2f |
  2c9dxp4-firefox/|
6c 69 62 2f 66 69 72 65 66 6f 78 2d 31 2e 34 2e |
  lib/firefox-1.4.|
31 0a 4d 52 48 4f 4d 45 3d 2f 6e 69 78 2f 30 6f |
  1.MRE_HOME=/nix/|
73 74 6f 72 65 2f 6a 31 35 33 68 6b 67 36 6e |
  store/0f9hrdwh3n|
64 33 6d 7a 35 63 71 6c 79 35 62 77 39 64 33 |
  d3mz5cqcncly5bw9|
...

Solution

▶ Compute hashes *modulo self-references*: when computing the final hash, replace every occurrence of the temporary hash by zeroes

▶ *Rewrite* occurrences of the temporary hash to the final hash

▶ Does this work? Yes!
So how does this help?

- A single derivation can now have different outputs.
- In particular substitutes can now be user-specific.
Example

Alice

- Gets `firefox.nix`
- Pulls from `evil.org`
- Runs `nix-env -i firefox`
  Selects substitute:
  `/nix/store/78k8w842kl8p...-firefox`
  Fake substitute is downloaded

Nix store

`/nix/store`
Example

Alice

▶ Gets firefox.nix
▶ Pulls from evil.org
▶ Runs nix-env -i firefox
  Selects substitute:
  /nix/store/78k8w842kl8p...-firefox
  Fake substitute is downloaded

http://evil.org/
Contains Trojan horse substitute
78k8w842kl8p...-firefox.nar.bz2.

Nix store

/nix/store
  ...
Example

Alice

- Gets firefox.nix
- Pulls from evil.org
- Runs nix-env -i firefox
  Selects substitute:
  /nix/store/78k8w842kl8p...-firefox
  Fake substitute is downloaded

Nix store

/nix/store

http://evil.org/
Contains Trojan horse substitute
78k8w842kl8p...-firefox.nar.bz2.
Example

Alice

- Gets `firefox.nix`
- Pulls from `evil.org`
- Runs `nix-env -i firefox`
  Selects substitute:
  `/nix/store/78k8w842kl8p...-firefox`
  Fake substitute is downloaded

Nix store

'/nix/store/78k8w842kl8p...-firefox'

- bin
- lib
- `libmozz.so`
- ...

http://evil.org/

Contains Trojan horse substitute
78k8w842kl8p...-firefox.nar.bz2.
Example

Alice
- Gets firefox.nix
- Pulls from evil.org
- Runs nix-env -i firefox
  Selects substitute:
  /nix/store/78k8w842kl8p...-firefox
  Fake substitute is downloaded

Bob
- Gets firefox.nix
- Pulls from good.org
- Runs nix-env -i firefox
  Selects substitute:
  /nix/store/j153hbg6n21c...-firefox
  Good substitute is downloaded

Nix store
- /nix/store/78k8w842kl8p...-firefox
  bin
  lib
    libmozz.so
    ...

http://evil.org/
- Contains Trojan horse substitute
  78k8w842kl8p...-firefox.nar.bz2.
Example

Alice
▶ Gets `firefox.nix`
▶ Pulls from `evil.org`
▶ Runs `nix-env -i firefox`
Selects substitute:
`/nix/store/78k8w842kl8p...-firefox`
Fake substitute is downloaded

Bob
▶ Gets `firefox.nix`
▶ Pulls from `good.org`
▶ Runs `nix-env -i firefox`
Selects substitute:
`/nix/store/j153hbg6n21c...-firefox`
Good substitute is downloaded

`http://evil.org/`
Contains Trojan horse substitute
`78k8w842kl8p...-firefox.nar.bz2`.

`http://good.org/`
Contains bona fide substitute
`j153hbg6n21c...-firefox.nar.bz2`. 
Example

**Bob**
- Gets *firefox.nix*
- Pulls from *good.org*
- Runs *nix-env -i firefox*
  Selects substitute:
  `/nix/store/j153hbg6n21c...-firefox`
  Good substitute is downloaded

**Alice**
- Gets *firefox.nix*
- Pulls from *evil.org*
- Runs *nix-env -i firefox*
  Selects substitute:
  `/nix/store/78k8w842kl8p...-firefox`
  Fake substitute is downloaded

**http://good.org/**
Contains bona fide substitute
*j153hbg6n21c...-firefox.nar.bz2.*

**http://evil.org/**
Contains Trojan horse substitute
*78k8w842kl8p...-firefox.nar.bz2.*

**Nix store**
```
/nix/store
 └── 78k8w842kl8p...-firefox
     └── lib
         └── libmozz.so
```
```
/nix/store
 └── j153hbg6n21c...-firefox
     └── bin
         └── firefox
     └── lib
         └── libmozz.so
```

```
/Nix store
```
```
/bin
```
```
/lib
```
```
/libmozz.so
```

How do we know which substitute to use for **firefox.nix**?

By computing the *output equivalence class*: a cryptographic hash of derivation attributes

- This is how the component's path was computed in the extensional model

Equivalence class + username is the key of the substitute mapping

**Example**

- Equivalence class for **firefox.nix** is
  
  `/nix/store/jjp9pi...-firefox`

- substitute`[(/nix/store/jjp9pi...-firefox, alice)]` =
  
  `(/nix/store/78k8w842kl8p...-firefox, ...url...)`

- substitute`[(/nix/store/jjp9pi...-firefox, bob)]` =
  
  `(/nix/store/j153hbg6n21c...-firefox, ...url...)`
Implementation aspect: Equivalence classes

How do we know which substitute to use for `firefox.nix`?

By computing the *output equivalence class*: a cryptographic hash of derivation attributes

- This is how the component's path was computed in the extensional model

Equivalence class + username is the key of the substitute mapping

Example

- Equivalence class for `firefox.nix` is
  `/nix/store/jjp9pi...-firefox`

- `substitute[('/nix/store/jjp9pi...-firefox, alice)] =
  (/nix/store/78k8w842kl8p...-firefox, ...url...)`

- `substitute[('/nix/store/jjp9pi...-firefox, bob)] =
  (/nix/store/j153hbg6n21c...-firefox, ...url...)`
Equivalence class collisions

Problem
When building, the inputs can contain multiple paths from the **same equivalence class**.
Equivalence class collisions (cont’d)

Solution

Rewrite *one* path from each equivalence class, then *rewrite* references.
Conclusions

- Main contribution: a package management system that allows any user to install software, with secure sharing between untrusted users.
- Content-addressable component stores allow binary components to be shared safely.
  - Hash rewriting is required to support self-referential components.
- It is possible to share locally built components safely.
- Transparent source/binary deployment can be done safely and selectively between mutually trusted users.