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

ASE 2005, Long Beach, CA

Eelco Dolstra
eelco@cs.uu.nl

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

November 10, 2005
Create a *package management system* that allows *any user* to install software.
Traditional Unix package managers

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

Windows, Mac OS X

- Everybody can install packages
- But there is no sharing (unless explicitly arranged)
Package management models

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

**Windows, Mac OS X**
- Everybody can install packages
- But there is no sharing (unless explicitly arranged)
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
Sharing

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

**Build attributes**

**Dependencies are built recursively**
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
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
User operations

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:

```
$ 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:

```bash
$ nix-push $(nix-instantiate firefox.nix) \ 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) \n   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) \ 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
Sharing in Nix

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
Alice

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

Builds it

Nix store

`/nix/store`
Sharing in Nix: Example

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
       └── 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!*

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!*

Nix store
```
/nix/store
  jjp9pi...-firefox
    bin
    firefox
    lib
    libxpcosm.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/x64bxp...-firefox`
  Builds it
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...-firefox`
  Builds it

Nix store
```
/nix/store
  jjp9pi...-firefox
    bin
    firefox
    lib
      libxpc...so
      libmoz...so
      ...
  x64b...-firefox
    bin
    firefox
    lib
      libxpc...so
      libmoz...so
      ...
```
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`
      - `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`
**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`

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

**Builder of `evil.nix`**
```
#!/bin/sh
cp trojan-horse
/nix/store/jjp9pi...-firefox/bin/firefox
```
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`  
      - `firefox`  
    - `lib`  
      - `libxpcom.so`  
      - `libmoz.so`  
      - `...`
- `01qr9w...-evil`  
  - `...`
Isolate builders

Run each build under a *unique user ID* (uid) so that they cannot interfere with each other.
**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

**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

**Nix store**

```
/nix/store
  ... ...
```

**http://evil.org/**

Contains Trojan horse substitute

`jjp9pi...-firefox.nar.bz2`. 
**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

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

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

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

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

Runs Firefox — 0wned!
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
      ...
```

*Note:* The Nix store diagram shows the structure of the Nix store with the Trojan horse substitute in place.
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`
    - `...`
Fake substitutes

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.
- Thus the trust relation must be established globally, for all users.
Solution: A content-addressable Nix store

- **Content-addressibility**: 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
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.
  \[ \text{$out = /nix/store/0f9hrdwh3nd3...-firefox} \]
- Run the builder
- Compute the hash $H$ over the output, e.g.
  \[ H = \text{j153hbg6n21c...} \]
- Rename the temporary path to \text{/nix/store/$H-name}, e.g.
  \[ /nix/store/j153hbg6n21c...-firefox \]
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.
   $\text{out} = /nix/store/0f9hrdwh3nd3...-firefox$

▶ 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

#! /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)

Solution

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

▶ \textit{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* occurrences of the temporary hash to the final hash

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

Solution

▸ Compute hashes \textit{modulo self-references}: when computing the final hash, replace every occurrence of the temporary hash by zeroes

▸ \textit{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 6a 31 35 33 68 62 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 6f 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 4f 4f 3d 2f 6e 69 78 2f 73 74 6f 72 65 2f 0f 9hrdwh3n|
  64 33 6d 7a 5f 6c 69 62 64 69 72 3d 2f 6e 69 78 35 63 71 63 6c 79 56 79 6b 79 64 70 78 36 6b 32 61 63 39 6d 78 70 34 2d 6f 72 65 66 6f 78 2f 21cb3ymykbydpx6k|
  64 33 6d 7a 5f 6c 69 62 64 69 72 3d 2f 6e 69 78 35 63 71 63 6c 79 56 79 6b 79 64 70 78 36 6b 32 61 63 39 6d 78 70 34 2d 6f 72 65 66 6f 78 2f 21cb3ymykbydpx6k|

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.*
<table>
<thead>
<tr>
<th>Alice</th>
<th>Nix store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets <code>firefox.nix</code></td>
<td><code>/nix/store</code></td>
</tr>
<tr>
<td>Pulls from <code>evil.org</code></td>
<td><code>___ ...</code></td>
</tr>
<tr>
<td>Runs <code>nix-env -i firefox</code></td>
<td></td>
</tr>
<tr>
<td>Selects substitute:</td>
<td></td>
</tr>
<tr>
<td><code>/nix/store/78k8w842kl8p...-firefox</code></td>
<td></td>
</tr>
<tr>
<td>Fake substitute is downloaded</td>
<td></td>
</tr>
</tbody>
</table>
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

Fake substitute is downloaded

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` ...

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

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

Nix store

```
/nix/store
    78k8w842kl8p...-firefox
        bin
        lib
        ...
```
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://evil.org`/
Contains Trojan horse substitute
`78k8w842kl8p...-firefox.nar.bz2`.

Nix store
```
/nix/store
  78k8w842kl8p...-firefox
     |
     bin
     |
     lib
     |
     ...!
```

Bin:
- Firefox executable
- Libs, e.g., `libmoz.so`
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`
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`

**Nix store**

```
/nix/store
  78k8w842kl8p...-firefox
    bin
    lib
      libmozz.so...
  j153hbg6n21c...-firefox
    bin
    firebox
    lib
      libmozz.so...
```

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

**http://good.org/**
- Contains bona fide substitute `j153hbg6n21c...-firefox.nar.bz2`
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
▶ We can share locally built components safely
▶ Transparent source/binary deployment can be done safely and selectively between mutually trusted users
▶ http://www.cs.uu.nl/groups/ST/Trace/Nix