IMPLEMENTATION STATUS

- Distr is freely available from ftp://ftp.eecs.tufts.edu/pub/distr
- revision: 2.0.0Alpha
  (not satisfied with specifications)
- handles UNIX files and links
  (directories easy given time)
- considering supporting NT
SCALABLE DISTRIBUTION ALLOWS

- scalable mistakes that disable networks very efficiently
  - network storms
  - rapid propagation of human errors
- scalable vulnerability to attack
  - cracking master cracks slaves
  - can be used for denial-of-service
AT WHAT COST?

- must create configurations for each kind of host and domain
- must manually configure a distribution topology for scalable use
- must bootstrap by distributing configuration files and Perl-5, perhaps with RDIST:)
ILLUSION AND REALITY

✗ illusion: PGP signing provides security
✔ reality: susceptible to replay attacks.
✗ illusion: difficult to write configurations
✔ reality: one basic file per host type
✗ illusion: it’d be easy to auto-configure
✔ reality: very hard problem
UNDERSTANDING IMPORT
SCALABILITY

import requests

responses

import requests
IMPLEMENTING SCALABILITY

- import.afterSuccess = sub {
  &some('export.initiate');
};
clients = ['foo','bar'];
follows each successful import with an export!

- export.before = sub {
  &some('import.initiate');
};
servers = ['foo','bar'];
queries servers for the correct versions before exporting to others!
**SCALABILITY**

if you told two friends, and they told two friends...

master server

intermediaries

slaves
SIMPLE HACKS

- `import.authentic = \&PGPauthentic;
signers = [‘Alva L. Couch’];
authenticates each file against detached PGP signature file.sig
- `import.before = sub {
  my $file = &some(‘import.file’);
  system("/usr/bin/ci -m ‘distr’ \n    $file >/dev/null 2>&1")/256==0;
};`
  implements local pre-distribution archiving.
- can be limited to **specific cases** by naming!
LOCAL CUSTOMIZATION

import = sub { # oversimplified to fit!
    if (&some('import.authentic')) {
        if (&some('import.before')) {
            if (&some('import.method')) {
                &some('import.afterSuccess');
            } else {
                &some('import.afterFailure');
            }
        } else {
            &some('import.afterDenial');
        }
    } else {
        &some('import.afterFailure');
    }
}

actually does the import

user ‘hooks’
USING DISTR

- distr -scopes mail.sendmail \ -tags export
calls distr on a master host to distribute files to a slave host

- distr -scopes mail.sendmail \ -tags import
calls distr on a slave host to request a file from a master host.
DISTR’S PROTOCOL

client initiates request

mail.sendmail.aliases.export.file

mail.sendmail.aliases.export.initiate

{‘tag’ => ‘import’,
 ‘scope’ => 
 ‘mail.sendmail.aliases’,
 ‘file’ => <embedded file> }

mail.sendmail.aliases.import

server responds to request
PARAMETER-PASSING

- `foo.import.file = '/foo';` is used by method `foo.import`
- `bar.import.file = '/bar';` is used by method `bar.import`
- both these methods are aliases for plain `import` (through inheritance)!
INHERITANCE

- **scope**: ‘where’ you are, e.g.,
  - `mail.sendmail.aliases`

- **tag**: ‘what’ you want, e.g., `import`

- use the first definition you find in the list:
  - `mail.sendmail.aliases.import`
  - `mail.sendmail.import`
  - `mail.import`
  - `import`

- Perl syntax: `&some('import')`
DISTR CONFIGURATION

mail.sendmail.aliases {
    import.file = '/usr/lib/aliases';
    import.afterSuccess = sub {
        system("'/usr/lib/newaliases \ 
            >/dev/null 2>&1")/256==0;
    };
};

- attributes can be arbitrary Perl-5 scalars, including function references
- missing details ‘filled in’ with inheritance
WHAT'S IN A NAME?

- `mail.sendmail.aliases` is the name of a (distributed) **object**
- `mail.sendmail.aliases.import.file = '/usr/lib/aliases';` specifies the target file.
- `mail.sendmail.aliases.import` is the **method** for importing that file
**DISTR**

- hosts are both **servers** and **clients**
  - server *distrd*: reacts to requests
  - client *distr*: makes requests
- hosts can be both **masters** and **slaves**
  - **master**: provider of information
  - **slave**: consumer of information
- **bidirectional**: master or slave initiates.
- slave machines must **agree** to updates!
  Masters **can’t force** slaves to comply!
**TYPICAL APPROACH (RDIST)**

mail:/usr/lib/aliases->(slave)

install /etc/mail/aliases

special "/usr/sbin/newaliases"

- requires a **master server**
- **unidirectional**: master-to-slave
- **platform-specific**
- master needs **root privileges** on slave
- this doesn’t exactly **encourage** cooperation between admins!
FILE DISTRIBUTION AND HETEROGENEITY

**Server**: master:
- provides file: /usr/lib/aliases

**Distribute**:
- ✓ transmit
- ✓ transform
- ✓ authenticate

**May have**:
- ✓ archive
- ✓ journal
- ✓ rollback
- ✓ different name
- ✓ different format
- ✓ different actions

**Client**: slave:
- receives file: /etc/mail/aliases
- executes: /usr/sbin/newaliases
AN ‘ANARCHIST’ VIEW

- replace a venerable and very mature tool (with a very young and strange one)!
- violate (almost all) software engineering and programming language principles!
- develop configuration maintenance architecture from the bottom up!
- redefine what is meant by ‘distribution’ (and perhaps even ‘scalable’)
TO GET ALONG, WE NEED:

- a **common language** for referring to things and actions
- the ability to **interpret** that language to make changes for the common good
- the ability to **limit changes** to those agreed upon by both parties
CHAOS OUT OF ORDER:
A SIMPLE, SCALABLE FILE DISTRIBUTION FACILITY
FOR “INTENTIONALLY HETEROGENEOUS” NETWORKS
-OR-
AN ANARCHISTS’ GUIDE
TO HETEROGENEOUS NETWORK CONFIGURATION MANAGEMENT

Alva L. Couch
Assoc. Prof. of EECS, Tufts University
Email: couch@eecs.tufts.edu
Web: http://www.cs.tufts.edu/~couch/