Introducing pyremctl:
A Python interface to remctl

Thomas L. Kula

2008 AFS and Kerberos Best Practices Workshop

One Minute remctl

- `http://www.eyrie.org/~eagle/software/remctl/`
- “Remctl allows you to remotely call a program and supply it with arguments, getting back any output and the exit status, authenticated with and encrypted by GSS-API Kerberos v5”

One Minute remctl

- Using the client you might call
  ```
  remctl mcketrick.tproa.net test i like tea
  ```

- And if configured like this:
  ```
  test ALL /path/to/remctl-test.sh ANYUSER
  ```

- remctld will execute the following command:
  ```
  /path/to/remctl-test.sh test i like tea
  ```

- And return to you the output (stdout and stderr) and exit status of that program

One Minute remctl

- Bonus features: the following environment variables are passed to the called program:
  - `REMOTE_USER`: the kerberos identity of the caller
  - `REMOTE_ADDR`: the IP address of the calling host
  - `REMOTE_HOST`: the name of the calling host
- You can either give remctld a list of principals allowed to run each command or use the above environment variables to make your decision

remctl bindings

- Using the remctl client is fine in some situations
- In other situations, you may want to embed it in something else
- Bindings for C and Perl (well tested) as well as Java (not as well tested)

Python bindings for remctl

- Current source: `http://kula.tproa.net/code/pyremctl/`
- Hand crafted (with love)
- Known to work with NetBSD, Python 2.4.3 and Remctl 2.11
- Can't think of any reason it shouldn't work with later stuff or on other operating systems

Installation

- Like any other python module
- Get and unpack the source
- `python setup.py build`
- `python setup.py install` (with appropriate permissions)
- Assumes remctl is already installed in a sane location

‘Simple’ interface

```
#!/usr/pkg/bin/python2.4

import remctl
import sys

host = ’mcketrick.tproa.net’
port = 4373
principal = ’host/mcketrick.tproa.net’
command = [ ’test’, ’i’, ’like’, ’tea’ ]

try:
    conn = remctl.remctl( host, port, principal, command )
```

```
except RemctlArgError:
    print "An invalid argument was supplied"
    sys.exit()
except RemctlProtocolError, e:
    print "Protocol error: " + e
    sys.exit()

if conn.stdout != None:
    print "Stdout: " + conn.stdout
if conn.stderr != None:
    print "Stderr: " + conn.stderr
print "Status: " + str(conn.status)
```

‘Simple’ interface

- Useful for just firing off a command

‘Complex’ interface

```
...

conn = remctl.Remctl()
conn.open( host, port, principal )

# Or: conn = remctl.Remctl( host, port, principal )
```

‘Complex’ interface

```
try:
    conn.command( command )
except RemctlArgError:
    sys.exit()
except RemctlProtocolError:
    sys.exit()
except RemctlError:
    sys.exit()
except RemctlNotOpened:
    sys.exit()
```

‘Complex’ interface

```
type, output, stream, status, error = conn.output()

while type != remctl.REMCTL_OUT_DONE:
    if type == remctl.REMCTL_OUT_OUTPUT:
        print "Stream " + str( stream ) +": " + output
    elif type == remctl.REMCTL_OUT_STATUS:
        print "Status: " + str( status )
    elif type == remctl.REMCTL_OUT_ERROR:
        print "Remctl error: " + error

    type, output, stream, status, error = conn.output()
```

‘Complex’ interface

- Useful for sending more than one command over a connection

Status

- Both interfaces seem to be working fine
- Not quite convinced this is idiomatic Python (My First Python/C Module)

Introducing pyremctl: A Python interface to remctl

Thomas L. Kula

`kula@tproa.net`

2008 AFS and Kerberos Best Practices Workshop

`http://kula.tproa.net/talks/afskbpw2008/kula-pyremctl.pdf`