

## Re: SSH pipe probelm

**Source:** <http://www.derkeiler.com/Newsgroups/comp.security.ssh/2004-11/0176.html>

---

**From:** Gary Tay Teng Teck ([garyttt\\_at\\_singnet.com.sg](mailto:garyttt_at_singnet.com.sg))

**Date:** 11/25/04

Date: Thu, 25 Nov 2004 22:00:14 +0800

Chris Nystrom wrote:

> *I have a graphical application and I want to use ssh as the backend,*  
> *and also have this transparent to the users for ease of use (ie, I*  
> *don't want to force users to install public keys).*  
>  
> *Everything works great, except for when ssh asks for the password.*  
> *Apparently ssh forces input from the TTY and redirecting STDIO does*  
> *not help. I have the password, but I do not want to force my*  
> *application users to go to the command line to type it in. How can I*  
> *get this password to the ssh process?*  
>  
> *Is there any work around to this issue, or has anyone already solved*  
> *this in some way?*  
>  
> *Please e-mail replies and I will summarize if there is interest.*  
>  
> *Thank you,*  
> *Chris*  
> *cnystrom@gmail.com*  
>  
> --  
>  
> *FYI - Here is my pipe code:*  
>  
> /\*  
> \* ssh-pipe.c  
> \*/  
>  
> #include <stdio.h> /\* standard I/O routines. \*/  
> #include <stdlib.h> /\* standard library routines. \*/  
> #include <unistd.h> /\* defines pipe(), amongst other things. \*/  
> #include <ctype.h> /\* defines isascii(), toupper(), and other \*/  
> /\* character manipulation routines. \*/  
> #include <string.h>  
>  
> #define READ 0  
> #define WRITE 1  
> #define ERROR 2

comp.security.ssh: Re: SSH pipe problm

```
>
> int make_pipe(char *username, char *hostname, char *program);
>
> void ssh_pipe(int input_pipe[], int output_pipe[], char *username,
> char *hostname, char *program);
>
> void puts_pipe(char *str);
> int gets_pipe(char *str, int max_len);
>
> /* 2 arrays to contain file descriptors for two pipes. */
> static int user_to_ssh[2];
> static int ssh_to_user[2];
>
> /* the main function: spawn off two processes */
> int
> make_pipe(char *username, char *hostname, char *program)
> {
> int pid; /* pid of child process, or 0, as returned via
> fork. */
> int rc; /* stores return values of various routines.
> */
>
> /* first, create one pipe. */
> rc = pipe(user_to_ssh);
> if (rc == -1) {
> perror("main: pipe user_to_ssh");
> exit(1);
> }
>
> /* then, create another pipe. */
> rc = pipe(ssh_to_user);
> if (rc == -1) {
> perror("main: pipe ssh_to_user");
> exit(1);
> }
>
> /* now fork off a child process, and set their handling routines.
> */
> pid = fork();
>
> switch (pid) {
> case -1: /* fork failed. */
> perror("main: fork");
> exit(1);
> case 0: /* inside child process. */
> ssh_pipe(user_to_ssh, ssh_to_user, username, hostname,
> program);
> /* NOT REACHED */
> default: /* inside parent process. */
> /* close unnecessary file descriptors */
> close(ssh_to_user[WRITE]); /* don't need to write to this
```

```

> pipe. */
> close(user_to_ssh[READ]); /* don't need to read from this
> pipe.*/
> }
>
> return 0; /* NOT REACHED */
> }
>
> void ssh_pipe(int input_pipe[], int output_pipe[], char *username,
> char *hostname, char *program)
> {
> char *newargs[6];
>
> /* first, close unnecessary file descriptors */
> close(input_pipe[WRITE]); /* we don't need to write to this pipe.
> */
> close(output_pipe[READ]); /* we don't need to read from this pipe.
> */
>
> /* dup so the pipe uses stdio */
> dup2(input_pipe[READ], READ);
> dup2(output_pipe[WRITE], WRITE);
>
> /* can close after dup, we just use stdio now */
> close(input_pipe[READ]);
> close(output_pipe[WRITE]);
>
> newargs[0] = "ssh";
> newargs[1] = "-l";
> newargs[2] = username;
> newargs[3] = hostname;
> newargs[4] = program;
> newargs[5] = 0;
>
> if( execvp ("ssh", newargs) )
> printf("exec returned non-zero value\n");
>
> exit(0);
> }
>
> void
> puts_pipe(char *str)
> {
> int i;
>
> i = strlen(str);
>
> // printf("OUT> %s\n", str);
>
> if(write(user_to_ssh[WRITE], str, i) == -1)
> perror("write");

```

```
> }  
>  
> int  
> gets_pipe(char *str, int max_len)  
> {  
> int i;  
>  
> if ((i = read(ssh_to_user[READ], str, max_len)) == -1)  
> perror("read");  
>  
> str[i] = '\0';  
>  
> // printf("IN> %s\n", str);  
>  
> return(strlen(str));  
> }
```

Maybe you could use "expect", I have never used it.

<http://expect.nist.gov/>