Security of Unix Pipes

Options

7 messages - Collapse all

David T. Ashley View profile More options Apr 30, 11:05 pm

I'm going to write an application that passes data between a parent and child process (bi-directionally) in pipes.

Are pipes secure so that other processes (even ones with the same UID/GID) can't eavesdrop on the information flowing between the two processes?

--
David T. Ashley (d...@e3ft.com)
http://www.e3ft.com (Consulting Home Page)
http://www.dtashley.com (Personal Home Page)
http://gpl.e3ft.com (GPL Publications and Projects)

Reply Reply to author Forward Rate this post:

Alan Curry View profile More options May 1, 1:32 am

In article <1I2dndIW1brgM6vbnZ2dnUVZ_h-vn...@giganews.com>,
David T. Ashley <d...@e3ft.com> wrote:

>I'm going to write an application that passes data between a parent and
>child process (bi-directionally) in pipes.

>Are pipes secure so that other processes (even ones with the same UID/GID)
>can't eavesdrop on the information flowing between the two processes?

No. Separate uids are the only real security barrier. When you inspect the memory of a process with the same uid, that's not called eavesdropping; it's called debugging.

--
Alan Curry
pac...@world.std.com

Logan Shaw View profile More options May 1, 1:43 am

David T. Ashley wrote:
> I'm going to write an application that passes data between a parent and
> child process (bi-directionally) in pipes.

> Are pipes secure so that other processes (even ones with the same UID/GID)
> can't eavesdrop on the information flowing between the two processes?

I'm not 100% sure, but it seems to me that no mechanism is safe from another process of the same UID, because a process of the same UID can do process tracing (ptrace() and stuff like that).

- Logan

Paul Pluzhnikov View profile More options May 1, 1:47 am

pac...@TheWorld.com (Alan Curry) writes:
> In article <1I2dndIW1brgM6vbnZ2dnUVZ_h-vn...@giganews.com>,
> David T. Ashley <d...@e3ft.com> wrote:
>>I'm going to write an application that passes data between a parent and
>>child process (bi-directionally) in pipes.

>>Are pipes secure so that other processes (even ones with the same UID/GID)
>>can't eavesdrop on the information flowing between the two processes?

> No. Separate uids are the only real security barrier. When you inspect the
> memory of a process with the same uid, that's not called eavesdropping; it's
> called debugging.

In addition to inspecting memory, many UNIX OSes provide a way to
inspect system call parameters via "truss" or "strace" utility,
and finding out what is being written to, or read from, pipe takes
no effort at all:

```
$ strace -s1024 -e trace=write /bin/echo "Super secret message written to pipe" |
  cat > /dev/null
write(1, "Super secret message written to pipe\n", 37) = 37
```

```
$ /bin/echo "Super secret message written to pipe" |
  strace -s1024 -e trace=read cat > /dev/null
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0...) = 512              # loader reads ELF header
read(0, "Super secret message written to pipe\n", 4096) = 37  # program reads the message
read(0, "", 4096)                       = 0                 # program gets EOF
```

If you want to hide this data, encrypt it with some stream cipher.
Parent can generate a random "session" key, which children will
inherit across fork.

Cheers,
--
In order to understand recursion you must first understand recursion.
Remove /-nsp/ for email.

Reply Reply to author Forward Rate this post:

Paul Pluzhnikov View profile More options May 1, 1:53 am

Paul Pluzhnikov <ppluzhnikov-...@charter.net> writes:
> If you want to hide this data, encrypt it with some stream cipher.

Perhaps I should clarify that: encrypting will make it non-trivial
(though still not terribly difficult for someone with a debugger
and assembly-level skills) to "listen in" on the communication,
which otherwise is trivially observable via 'strace' or 'truss'.

Cheers,
--
In order to understand recursion you must first understand recursion.
Remove /-nsp/ for email.

Reply Reply to author Forward Rate this post:

Chris McDonald View profile More options May 1, 7:49 am

Paul Pluzhnikov <ppluzhnikov-...@charter.net> writes:
>Paul Pluzhnikov <ppluzhnikov-...@charter.net> writes:
>> If you want to hide this data, encrypt it with some stream cipher.
>Perhaps I should clarify that: encrypting will make it non-trivial
>(though still not terribly difficult for someone with a debugger
>and assembly-level skills) to "listen in" on the communication,
>which otherwise is trivially observable via 'strace' or 'truss'.

Your original question asked if the *pipes* were secure, to which I'd probably say 'yes'.

Most of the replies received have discussed the ability to interogate the parent and child processes, watching what is written into, and what read from, the pipe. The ability to do this does not reflect on the security of the pipe itself. Similarly, talk of performing encryption, just before writing to the pipe, seems misplaced. Any of the stated debugging/inspection techniques will just examine the data before it's encrypted - again, nothing to do with the pipe.

As the pipe (the buffer) resides in the kernel, potential eavesdroppers will require access to that memory to violate the security of the pipe, itself. As with most things security related - what is the threat that you're trying to deny?

_____
Dr Chris McDonald E: c...@csse.uwa.edu.au
Computer Science & Software Engineering W: http://www.csse.uwa.edu.au/~chris
The University of Western Australia, M002 T: +618 6488 2533
Crawley, Western Australia, 6009 F: +618 6488 1089

Paul Pluzhnikov View profile More options May 1, 10:50 am

Chris McDonald <c...@csse.uwa.edu.au> writes:
> Paul Pluzhnikov <ppluzhnikov-...@charter.net> writes:

>>Perhaps I should clarify that: encrypting will make it non-trivial
...
> Your original question

It wasn't *my* question.

> asked if the *pipes* were secure, to which I'd
> probably say 'yes'.

Without specifying 'secure against what?' your answer is meaningless.
In the context of original question:

>>Are pipes secure so that other processes ...
>>can't eavesdrop on the information flowing between the two processes?

your answer is probably wrong -- pipes themselves do not prevent
other processes from eavesdropping on the communication, because the
info is trivially observed before it "enters the pipe" and after it
"leaves the pipe".

> Similarly, talk of performing encryption,
> just before writing to the pipe, seems misplaced.

It doesn't seem misplaced to me -- it answers the question "How
can I prevent (or make it more difficult) other process from
eavesdropping on my communication via pipe?"

> As the pipe (the buffer) resides in the kernel, potential eavesdroppers
> will require access to that memory to violate the security of the pipe,
> itself.

Yes, the pipe *itself* is "secure" against observing the data in it.
This does not make eavesdropping on the info going through the pipe
impossible, or even difficult.

> As with most things security related - what is the threat that
> you're trying to deny?

Seems pretty clearly stated in the original message.

Cheers,
--
In order to understand recursion you must first understand recursion.
Remove /-nsp/ for email.

Reply Reply to author Forward Rate this post: ☆☆☆☆☆

End of messages

« Back to Discussions « Newer topic Older topic »

Security of Unix Pipes (with Application Details)

Options

13 messages - Collapse all

David T. Ashley View profile

More options May 1, 10:44 am

Thanks for the helpful replies on the other thread. Essentially it was pointed out that a process with the same UID as the ones writing/reading the pipes would be able to examine the memory of the pipe writers/readers and figure out what data had flowed.

Here is my application (and I'd be grateful for any other advice).

I'm incorporating cryptographic FOBs into my PHP-based database application to enhance login security.

A cryptographic FOB is an electronic device about the size of a keychain that either displays sequential numbers to be used as one-time passwords or else can allow you to enter a long number (the challenge) and will generate a number in response (the reply) to be used as a password.

Here is a page with some photos of similar devices:

http://en.wikipedia.org/wiki/Security_token

The FOB contains a key (typically a 192-bit AES key) that is used for both sequential one-time password generation and for challenge-reply.

Compromise of the FOB key is equivalent to breaking security (because with the key one can emulate the FOB). The theory of operation of the FOB is that one can observe a great many sequential numbers or challenge-response cycles and the mathematical framework does not exist to use this information to guess what the FOB will display next or how it will respond to a different challenge, or to reverse-engineer the key.

The manufacturer of the FOB provides a Linux shared library to use in authenticating the FOB. The library is essentially a cryptographic math library. To determine, for example, what the FOB's response to a certain challenge should be, one would make a call into the library with the FOB key and the challenge, and the function will return what the FOB should answer.

I am not able to call a shared library directly from PHP. Instead, I have to write a C program that is called (i.e. exec'd or similar) from PHP and which uses the shared library.

Because the FOB key is one of the parameters that must be used with the shared library, it must also be passed from PHP to the compiled C program. Because the FOB key is so sensitive, the question is how to pass it from PHP to the compiled program securely.

Passing the information on the command line is clearly not secure, because program names and command-line parameters are world-visible on a Unix system.

However, I was thinking that I could use the PHP proc_open() function:

http://us.php.net/manual/en/function.proc-open.php

to pass the information to the compiled C program's stdin and get information back from stdout securely (without others being able to eavesdrop).

If I've read everyone's posts correctly, the only security hole is if a potential attacker could launch another process with the same UID.

I guess also I'd need to wipe memory before the compiled C program terminates to get rid of any trace of the sensitive information (otherwise the memory might be discovered by other processes later).

Any other suggestions or thoughts?

Thanks
--
David T. Ashley (d...@e3ft.com)
http://www.e3ft.com (Consulting Home Page)
http://www.dtashley.com (Personal Home Page)
http://gpl.e3ft.com (GPL Publications and Projects)

Reply Reply to author Forward Rate this post:

Toby A Inkster View profile More options May 1, 11:35 am

David T. Ashley wrote:
> I am not able to call a shared library directly from PHP. Instead, I have

> to write a C program that is called (i.e. exec'd or similar) from PHP and
> which uses the shared library.

Have you considered writing an extension to PHP? This is a compiled (C)
shared object that could interface between your PHP script and the library
in question.

I've not written one before, but I'm told they're easier than you'd expect.

--
Toby A Inkster BSc (Hons) ARCS
http://tobyinkster.co.uk/
Geek of ~ HTML/SQL/Perl/PHP/Python/Apache/Linux

Reply Reply to author Forward Rate this post:

David T. Ashley View profile More options May 1, 12:32 pm

"Toby A Inkster" <usenet200...@tobyinkster.co.uk> wrote in message
news:fb3kg4-7eq.ln1@ophelia.g5n.co.uk...

> David T. Ashley wrote:

>> I am not able to call a shared library directly from PHP. Instead, I
>> have
>> to write a C program that is called (i.e. exec'd or similar) from PHP and
>> which uses the shared library.

> Have you considered writing an extension to PHP? This is a compiled (C)
> shared object that could interface between your PHP script and the library
> in question.

> I've not written one before, but I'm told they're easier than you'd
> expect.

This is an interesting idea. The only potential barrier in my case is that
I run Red Hat Enterprise Linux and I receive my PHP as packages from Red Hat
(which is great for me).

Does this involve recompiling PHP? Or does it involve some external work
that links to PHP? In other words, could I continue to use the pre-built
PHP packages from Red Hat?

Thanks.

Reply Reply to author Forward Rate this post:

Rainer Weikusat View profile More options May 1, 1:04 pm

"David T. Ashley" <d...@e3ft.com> writes:

[...]

- Show quoted text -

You could try something simple, like writing the key to a file only readable by someone with the 'correct' UID and pass the name of the file to the program via commandline argument.

Reply Reply to author Forward Rate this post:

Paul Pluzhnikov View profile More options May 1, 1:01 pm

"David T. Ashley" <d...@e3ft.com> writes:

> I am not able to call a shared library directly from PHP.

You should be able to write a PHP extension (another shared library) which will wrap the vendor-supplied library and provide an interface that PHP expects. You should be able to load that extension into unmodified PHP packages you get from RedHat.

> Because the FOB key is one of the parameters that must be used with the
> shared library, it must also be passed from PHP to the compiled C program.
> Because the FOB key is so sensitive, the question is how to pass it from PHP
> to the compiled program securely.

There is no method that will be secure against debugger (even the PHP extension is prone to debugger discovery of the secret).

If you ignore the debugger, encrypting (via plain XOR) the FOB key with another key, which is known to your compiled C program and to your PHP module is the answer. You can then pass the encrypted key any way you want: on command line, via pipe, through the environment variable, in shared memory, etc. etc.

> Passing the information on the command line is clearly not secure, because
> program names and command-line parameters are world-visible on a Unix
> system.

And so are environment variables, and so are pipes, and so are files. Any communication between your PHP process and your compiled C

program is very easy to "sniff" from another process with the same UID.

> However, I was thinking that I could use the PHP proc_open() function:

> http://us.php.net/manual/en/function.proc-open.php

> to pass the information to the compiled C program's stdin and get
> information back from stdout securely (without others being able to
> eavesdrop).

Other processes with the same UID (or root) will be able to trivially
eavesdrop (as we told you before).

> I guess also I'd need to wipe memory before the compiled C program
> terminates to get rid of any trace of the sensitive information (otherwise
> the memory might be discovered by other processes later).

I believe this attack is impossible on any modern UNIX -- the OS
will not give "dirty" RAM to another process.

Cheers,
--
In order to understand recursion you must first understand recursion.
Remove /-nsp/ for email.

Reply Reply to author Forward Rate this post:

David T. Ashley View profile More options May 1, 2:19 pm

"Rainer Weikusat" <rweiku...@mssgmbh.com> wrote in message

news:87tzuwxypg.fsf@fever.mssgmbh.com...

> "David T. Ashley" <d...@e3ft.com> writes:

> [...]

>> However, I was thinking that I could use the PHP proc_open() function:

>> http://us.php.net/manual/en/function.proc-open.php

>> to pass the information to the compiled C program's stdin and get
>> information back from stdout securely (without others being able to
>> eavesdrop).

> You could try something simple, like writing the key to a file only
> readable by someone with the 'correct' UID and pass the name of the
> file to the program via commandline argument.

Yeah, this may be simplest of all. Now that everyone has shattered my vision of pipes as secure, this is possible also. If you have a UID/GID adequate to read the file, then you have a UID/GID adequate to eavesdrop on pipes as well.

Reply Reply to author Forward Rate this post:

David T. Ashley View profile More options May 1, 2:25 pm

"Paul Pluzhnikov" <ppluzhnikov-...@charter.net> wrote in message

news:m3abwo5vht.fsf@somewhere.in.california.localhost...

> "David T. Ashley" <d...@e3ft.com> writes:

>> I am not able to call a shared library directly from PHP.

> You should be able to write a PHP extension (another shared library)
> which will wrap the vendor-supplied library and provide an interface
> that PHP expects. You should be able to load that extension into
> unmodified PHP packages you get from RedHat.

I'll look into this. If anybody wants to throw me a URL ...

>> Because the FOB key is one of the parameters that must be used with the
>> shared library, it must also be passed from PHP to the compiled C
>> program.
>> Because the FOB key is so sensitive, the question is how to pass it from
>> PHP
>> to the compiled program securely.

> There is no method that will be secure against debugger (even the
> PHP extension is prone to debugger discovery of the secret).

> If you ignore the debugger, encrypting (via plain XOR) the FOB key
> with another key, which is known to your compiled C program and to
> your PHP module is the answer. You can then pass the encrypted key
> any way you want: on command line, via pipe, through the environment
> variable, in shared memory, etc. etc.

This is actually a great idea! If the executable program that links to the FOB maker's shared library has permissions set so that only the Apache/PHP

UID can access it at all, then the command line is just as secure as any other mechanism if a secret is shared. If the distribution of the keys is random, the XOR mechanism will also provide a random distribution ... there would be absolutely NO information gleanable from watching command-line arguments.

>> I guess also I'd need to wipe memory before the compiled C program
>> terminates to get rid of any trace of the sensitive information
>> (otherwise
>> the memory might be discovered by other processes later).

> I believe this attack is impossible on any modern UNIX -- the OS
> will not give "dirty" RAM to another process.

This is something I did not know. I just assumed that if one does malloc() or similar or pokes around via other mechanisms in memory that won't cause a segfault, the memory contents would be whatever was left over from whatever ran before. I was not aware that modern Unix system wiped memory before giving it to another process.

--
David T. Ashley (d...@e3ft.com)
http://www.e3ft.com (Consulting Home Page)
http://www.dtashley.com (Personal Home Page)
http://gpl.e3ft.com (GPL Publications and Projects)

Reply Reply to author Forward Rate this post:

Eric Sosman View profile More options May 1, 3:20 pm

David T. Ashley wrote On 05/01/07 14:19,:

- Show quoted text -

Yeah, but opening and reading a named file in the file system is noticeably easier than rummaging around in the address space of a process. Can be done more surreptitiously, too: I just have a little program that sits around and waits for files to appear, then opens and reads them as promptly as it can. Yes, some of them will escape my notice -- but I'll get a steady trickle.

Meanwhile, attaching a debugger to a process that's delivering a service has an unfortunate tendency to slow down the service, or even to pause it for macroscopic time. (The impact of truss and such isn't too bad, but

if you encrypt the traffic on the pipe the attacker is
going to need more than truss can reveal.) When your help
desk phones start ringing with folks complaining that they
can't log in, somebody's likely to take a look at what's
wrong on the authentication server, and there's the attacker
running gdb ...

As an attacker (not in real life, I hasten to add), I'd
feel lots less exposed snooping in the file system than I
would hunched over a gdb session.

Besides: I don't think I'd bother with your pipes or
temp files or shared memory or whatever else, at least not
for my first attempt. No, I'd go after the database with
which you associate user IDs to FOB keys. At least, that's
where I'd begin, until and unless it proved sufficiently
armored against my depraved schemes.

--
Eric.Sos...@sun.com

Reply Reply to author Forward Rate this post:

Eric Sosman View profile More options May 1, 3:35 pm

David T. Ashley wrote On 05/01/07 14:25,:

> This is actually a great idea! If the executable program that links to the
> FOB maker's shared library has permissions set so that only the Apache/PHP
> UID can access it at all, then the command line is just as secure as any
> other mechanism if a secret is shared. If the distribution of the keys is
> random, the XOR mechanism will also provide a random distribution ... there
> would be absolutely NO information gleanable from watching command-line
> arguments.

You don't even need the shared secret in advance.
Look up "Diffie-Hellman key exchange," a method by which
two parties can agree on a dynamically-created shared
secret while communicating over an insecure channel.
The attacker can see every message that passes over the
channel (in some variations, the attacker can even modify
the messages), and *still* can't discover the key the
two parties eventually agree on.

It seems to me you'd do well to browse the FAQs of
some of the crypto newsgroups -- you'd probably get some

useful pointers and some even-more-useful warnings ...

--
Eric.Sos...@sun.com

Reply Reply to author Forward Rate this post:

David T. Ashley View profile More options May 1, 4:15 pm

"Eric Sosman" <Eric.Sos...@sun.com> wrote in message

news:1178048111.904440@news1nwk...

- Show quoted text -

In another life I need to become a number theorist, just to get an appreciation for that stuff.

I've written code, for example, to apply the Miller-Rabin primality test. It just amazes me that it is a relatively cheap operation to prove that a number is not prime, but potentially a very expensive operation to actually factor it. On some level it amazes me that you can pick two very large integers, determine to a high level of certainty that they are prime, then use that as the basis for an ecryption system.

Diffie-Hellman is a miracle.

AES, DES, MD5, and SHA1 are similar miracles. I have no practical idea why they have the properties they have. To me they are all just bit-scrambling code.

--
David T. Ashley (d...@e3ft.com)
http://www.e3ft.com (Consulting Home Page)
http://www.dtashley.com (Personal Home Page)
http://gpl.e3ft.com (GPL Publications and Projects)

Reply Reply to author Forward Rate this post:

Chung Leong View profile More options May 1, 4:51 pm

On May 1, 4:44 pm, "David T. Ashley" <d...@e3ft.com> wrote:

> I guess also I'd need to wipe memory before the compiled C program
> terminates to get rid of any trace of the sensitive information (otherwise

> the memory might be discovered by other processes later).

If the FOB key passes through PHP, then it'll linger in memory in its address space. I don't see any effective way you can wipe it in PHP.

Reply Reply to author Forward Rate this post:

Rainer Weikusat View profile More options May 2, 3:45 am

- Show quoted text -

[...]

> As an attacker (not in real life, I hasten to add), I'd
> feel lots less exposed snooping in the file system than I
> would hunched over a gdb session.

If somebody is running processes with either your UID (or a more priviledged one) on the machine that tries to 'attack' you, you are toast. There is no need for a 'gdb session', just write a program that attaches to the to-be-attacked process, use PTRACE_SYSCALL (Linux) to stop it after each syscall and modifiy the running image to your hearts content (like setting up a 'fake pipe' through the eavesdropping program).

"I have taken great pains to be reasonably safe from stupid attackers" doesn't sound that good.

Reply Reply to author Forward Rate this post:

Toby A Inkster View profile More options May 2, 4:00 am

David T. Ashley wrote:
> Does this involve recompiling PHP? Or does it involve some external work
> that links to PHP? In other words, could I continue to use the pre-built
> PHP packages from Red Hat?

I imagine that if you download and install the source RPM for PHP from Red Hat, then you should be able to compile your extension against that source code, without having to recompile PHP itself.

Certainly I have downloaded and compiled PHP extensions from PECL and I didn't need to recompile the whole of PHP.

--
Toby A Inkster BSc (Hons) ARCS
http://tobyinkster.co.uk/
Geek of ~ HTML/SQL/Perl/PHP/Python/Apache/Linux