Config files for your script

General

For this task, you don't have to write large parser routines (unless you want it 100% secure or you want a special file syntax) - you can use the Bash source command. The file to be sourced should be formated in key="value" format, otherwise bash will try to interpret commands:

```
#!/bin/bash
echo "Reading config...." >&2
source /etc/cool.cfg
echo "Config for the username: $cool_username" >&2
echo "Config for the target host: $cool_host" >&2
```

So, where do these variables come from? If everything works fine, they are defined in /etc/cool.cfg which is a file that's sourced into the current script or shell. Note: this is **not** the same as executing this file as a script! The sourced file most likely contains something like:

```
cool_username="guest"
cool_host="foo.example.com"
```

These are normal statements understood by Bash, nothing special. Of course (and, a big disadvantage under normal circumstances) the sourced file can contain **everything** that Bash understands, including malicious code!

`source` command also is available under the name `.` (dot). The usage of the dot is identical:

```
#!/bin/bash
echo "Reading config...." >&2
. /etc/cool.cfg #note the space between the dot and the leading slash of /etc.cfg
echo "Config for the username: $cool_username" >&2
echo "Config for the target host: $cool_host" >&2
```

Per-user configs

There's also a way to provide a system-wide config file in /etc and a custom config in ~/(user's home) to override system-wide defaults. In the following example, the if/then construct is used to check for the existance of a user-specific config:

```
#!/bin/bash
echo "Reading system-wide config...." >&2
. /etc/cool.cfg
if [ -r ~/.coolrc ]; then
  echo "Reading user config...." >&2
  . ~/.coolrc
fi
```

Secure it

As mentioned earlier, the sourced file can contain anything a Bash script can. Essentially, it an included Bash script. That creates security issues. A malicicios person can "execute" arbitrary code when your script is sourcing its config file. You might want to allow only constructs in the form `NAME=VALUE` in that file (variable assignment syntax) and maybe comments (though technically, comments are unimportant). Imagine the following "config file", containing some malicious code:

```
# cool config file for my even cooler script
username=god_only_knows
hostname=www.example.com
password=secret ; echo rm -rf ~/*
parameter=foobar && echo "You've bene pwned!";
# hey look, weird code follows...
echo "I am the skull virus..."
echo rm -fr ~/*
mailto=netadmin@example.com
```

You don't want these `echo` -commands (which could be any other commands!) to be executed. One way to be a bit safer is to filter only the constructs you want, write the filtered results to a new file and source the new file. We also need to be sure something nefarious hasn't been added to the end of one of our name=value parameters, perhaps using ; or && command separators. In those cases, perhaps it is simplest to just ignore the line entirely. Egrep (`grep -E`) will help us here, it filters by description:

```
#!/bin/bash
configfile='/etc/cool.cfg'
configfile_secured='/tmp/cool.cfg'

# check if the file contains something we don't want
if egrep -q -v '^#|^[^ ]*=[^;]*' "$configfile"; then
  echo "Config file is unclean, cleaning it..." >&2
  # filter the original to a new file
  egrep '^#|^[^ ]*=[^;&]*'  "$configfile" > "$configfile_secured"
  configfile="$configfile_secured"
fi

# now source it, either the original or the filtered variant
source "$configfile"
```

To make clear what it does: egrep checks if the file contains something we don't want, if yes, egrep filters the file and writes the filtered contents to a new file. If done, the original file name is changed to the name stored in the variable `configfile`. The file named by that variable is sourced, as if it were the original file.

This filter allows only `NAME=VALUE` and comments in the file, but it doesn't prevent all methods of code execution. I will address that later.

Discussion

Iñigo (http://poisonbit.wordpress.com), 2010/09/05 14:55 ()

Please take $() in consideration

```
inigo@crono:~/tmp/cfg$ cat test.cfg
name=fooo
address="fooo@fooo.org $(echo $(ls))"
```

Using the article filter:

```
inigo@crono:~/tmp/cfg$ bash cfg.sh
username: fooo
address:  fooo@fooo.org cfg.sh test.cfg
```

The $(echo $(ls)) (or any other command) may get executed.

The same could be done with \`...\`

Jan Schampera, 2010/09/13 04:34 ()

Yes, you're absolutely right.

I wonder, before tinkering something else into the code, if it makes sense to take this article offline. The way is nice, simple, effective, but unusable, no matter what you do. It is and stays a workaround to not have to code too much for a config file system.

Reinier Boon, 2011/01/19 15:40 ()

I succesfully used a construction like

```
  MAIL_TO=`perl -ne '/^\s*MAIL_TO\s*=\s*([\s\w@\d,\._-]+)/ && do
{print $1; exit;}' $CONFIG_FILE`
```

to read the users to mail to from a config file...

It depends a bit on what exactly will be in your config variable, you may have to tweak the regexp.

Amine MECIFI (http://www.linuxhope.com), 2011/11/26 11:03 ()

**Please don't delete this tip. It's a good one!

Security wise, if someone is able to temper with the config file, he will be also able to edit the shell script file in first place.

If you are really concerned by the security, you can do the following:

- once you are satisfied that your config file is complete, do a md5sum and hard code it into the shell script. So the script will check the md5sum of the conf file every time it calls it. - The problem with this, is if you change something in the conf file, you need to issue an new md5sum and place in the shell. - You can be more aggressive (paranoid?) but splitting the config file in 2 or more files, test the md5sum of each of them. Then you also test the md5sum of the whole file and you compare everything. This is just a provision for the case where someone comes up with an altered config file but with the same md5sum (theoretically possible).

Jan Schampera, 2011/11/29 06:54 ()

An attacker who can edit the config isn't necessarily able to edit the script, just think of configs in dot-file in the home directory, or the-like.

You are also right, yes, but a warning is absolutely needed here 😉

ano (http://www.ano.de), 2012/07/27 22:50 ()

there are many ways to break through the filter

jhv@hyperion:~/Desktop/test$ cat conf #this is the config-file test1=ssss && touch yougotowned0 test2=aaa;touch yougotowned1 test2=`./virus` ./virus;k=www 1test2=false||touch yougotowned2 test2=false touch yougotowned3 test3=$(./virus) ./virus\=a gg=./virus $gg;h=hh

jhv@hyperion:~/Desktop/test$ ls -l insgesamt 20 -rwxr-xr-x 1 jhv jhv 38 2012-07-28 00:26 clear -rwxr-xr-x 1 jhv jhv 233 2012-07-28 00:42 conf -rwxr-xr-x 1 jhv jhv 449 2012-07-28 00:40 config_loader -rwxr-xr-x 1 jhv jhv 535 2012-07-28 00:22 org -rwxr-xr-x 1 jhv jhv 45 2012-07-28 00:22 virus lrwxrwxrwx 1 jhv jhv 28 2012-07-28 00:29 virus=a → /home/jhv/Desktop/test/virus jhv@hyperion:~/Desktop/test$ #this is the "virus" jhv@hyperion:~/Desktop/test$ cat virus #!/bin/bash echo boom >&2 touch owned$RANDOM jhv@hyperion:~/Desktop/test$ ./config_loader Config file is unclean, cleaning it… boom boom /tmp/cool.cfg: Zeile 6: 1test2=false: Kommando nicht gefunden. boom boom boom jhv@hyperion:~/Desktop/test$ ls clear org owned14191 virus yougotowned1 conf owned10656 owned15845 virus=a yougotowned2 config_loader owned10851 owned5682 yougotowned0 yougotowned3 jhv@hyperion:~/Desktop/test$

Fahmy MF, 2012/08/12 06:52 ()

Hi, I need to configure build for each and every test environment with the specific set of file in the build relating to environment specific configuration, for example as bellow

File name - audit.properties

PROVIDER_URL=http://10.10.10.10:8080,11.11.11.11:8082 (http://10.10.10.10:8080,11.11.11.11:8082)

database.username=username database.psswd=password

Currently I am configuring manually keeping a document saved all the configurations against the test environment and configuration before deploy the build. What I am require is I need to get a shell script to configure the properties/configuration reading the environment specific configuration from a particular file which contains which file need to configure and which properties need to configure and the what is a value to be enter. then it would be easy to maintain also, since those values are frequently changing, Please help me out on this

Thank you in advance, Faz m Fah

Joe (http://sourceforge.net/projects/duplexpr/), 2013/01/18 06:24 (), 2014/10/06 04:30 ()

This is a variation on the methods above, but using sed that I am working on now. It's far from bullet proof, but it may give you some ideas:

```
  ## Strip all comment, null, and blank lines
  ## Strip all lines with shell nasties in them
  ##   (won't catch translated values like 0x24)
  ## Then pass only those lines that start with a valid parameter
name
  ## into a work file
  < "${PARAM_FILE}" sed -rn -e '/^#/d
  /^ /d
  /^$/d
  /[$`;><{}%|&!]/d
 /^DEST_DIR=|^DEST_LABEL=|^DRYRUN=|^MOUNT1=|^MOUNT2=|^NAME=|^PRUN
E=|^SOURCE_DIR=|^SOURCE_LABEL=|^TASK_NAME=|^UNMOUNT_SOURCE=/p'
> "${PARAM_WORK}"

... a few more edits ...

source "${PARAM_WORK}"
```

Tom, 2013/02/19 13:27 ()

If you want to make it a little bit more secure set the config file to 600 permission and the owner to the script executor. Then check in your script if owner and permission is ok. Still not perfect, i know

Code at random (http://codeatrandom.blogspot.com/), 2013/05/15 03:25 (), 2014/10/06 04:29 ()

Thanks for the great article, it helped me a lot to write my script! If I do concern about security, what I don't always do whit private scripts, I'm a bit more paranoid and the above explained approaches are not enough for me as they violate two very fundamental concepts in security:

- don't use black lists, use white lists (you can never be sure to catch all possibilities) - don't try to fix malicious input, abort instead and raise a error (otherwise there might be greater harm and if some thing fishy is going on, you should be notified about that)

(Of course there are exceptions for this rules, but not in this example.)

My approach to this is:

```
#!/bin/bash

c_file=config_file_test

unknown=`cat $c_file | grep -Evi "^(#.*|[a-z]*='[a-z0-9 ]*')$"`
if [ -n "$unknown" ]; then
 echo "Error in config file. Not allowed lines:"
 echo $unknown
 exit 1
fi
source $c_file
echo "File Loaded"
```

It is a quiet paranoid version, allowing only lines start whit a # as comments and variable declarations like "hello='world 88'". If you have stuff like URL's in your config file, you can add the needed characters (:/?#&) to the regex. There should be no escape from the string (or did I overlook some thing?). This way has as well the advantage that it doesn't need a temporary file.

Hope that helps some one to 😉

Sirat, 2013/07/14 09:42 (), 2014/10/06 04:31 ()

How about using whitelisting instead and rejecting the file if something turns out wrong:

```
#!/bin/bash

CONFIG_PATH='./bashconfig.conf'
# commented lines, empty lines und lines of the from choose_ANYNA
ME='any.:Value' are valid
CONFIG_SYNTAX="^\s*#|^\s*$|^[a-zA-Z_]+='[^']*'$"

# check if the file contains something we don't want
if egrep -q -v "${CONFIG_SYNTAX}" "$CONFIG_PATH"; then
  echo "Error parsing config file ${CONFIG_PATH}." >&2
  echo "The following lines in the configfile do not fit the synt
ax:" >&2
  egrep -vn "${CONFIG_SYNTAX}" "$CONFIG_PATH"
  exit 5
fi

# otherwise go on and source it:
source "${CONFIG_PATH}"
```

Jimmy, 2014/04/07 19:53 (), 2014/10/06 04:32 ()

What about this?

```
cat /dev/null > /tmp/rc.conf~

sed '/^ *#/d;/^[:space:]*$/d;s/=/ /;' < "/etc/rc.conf" | while re
ad key val
do
        #verify here
        #remove all unwanted
        val=$(echo "$val" | sed 's/[^[:alnum:]\.]//g')
        str="$key=\"$val\""
        echo "$str" >> /tmp/rc.conf~
done

. /tmp/rc.conf~
```

This should remove the most evil things…

Michael Grünewald (https://plus.google.com/u/0/+MichaelLeBarbierGrünewald/about), 2015/06/05 12:41 ()

As described in my blog article Configuration files for shell scripts (http://unix-workstation.blogspot.de/2015/06/configuration-files-for-shell-scripts.html) it is easy to use a sed script to convert INI-style configuration file to a tabular format which can be processed with awk or read, for instance.

howto/conffile.txt Last modified: 2015/08/08 16:00 by bill_thomson

This site is supported by Performing Databases - your experts for database administration

Bash Hackers Wiki