SANS Penetration Testing

06 Jun **2012**
https://pen-testing.sans.org/blog/2012/06/06/escaping-restricted-linux-shells

Escaping Restricted Linux Shells

6 comments Posted by eskoudis
Filed under Linux, Shell Fu

[Editor's Note: On the GPWN mailing list for SANS Pen Test Course Alumni a few months ago, we had a nice, lively discussion about techniques penetration testers and ethical hackers could use to escape a restricted shell environment. A lot of nifty techniques were offered in what amounted to an interactive brainstorming session on the list. Doug Stilwell offered to write an article based on the discussion and his own experience. I really like what he's come up with, and I think it'll be a handy reference for folks who find themselves facing a restricted shell in a pen test and need to get deeper access into the target system. Thanks for the cool article, Doug! -Ed.]

By Doug Stilwell

Introduction

Last year I was approached by a systems engineer and he offered me a steak dinner if I could escape the restricted shell he had set up on a Linux server. The restricted shell was being created due to a request from the development group needing access to certain servers for troubleshooting, and he wanted to restrict what they could do. I don't know about most of you, but a challenge alone from one of my colleagues is more than enough to get me going, but a steak dinner? Where do I sign?! The only thing better would have been a case of beer! I wasn't very familiar with restricted shells so off to Google I went for some assistance. It didn't take long before I noticed there was a lack of information on the topic. Information was scattered and most resources were not very descriptive. In addition to internet research, I also consulted the SANS GPWN e-mail distribution list to see if anybody had any cool tricks. It turns out, they did, but the consensus was that they'd like to see an article published covering this topic.

Scope

The purpose of this article is to create a better resource for penetration testers to assist them when confronted with a restricted shell. In no way will this be an exhaustive account of all techniques, but instead, I'm going to cite several of the most applicable and effective techniques in this handy reference document. This article is not going to cover defending restricted shell attacks, focus on specific flavors of Linux, nor is it going to cover every restricted shell since they can be configured many ways. Therefore, command line syntax may differ depending on the version of Linux you're familiar with. I will focus mainly on the techniques themselves. Of course, if you find a way to exploit a running process to escalate your privileges, then there is no need to escape the shell.

A Restricted Shell?What Is It?

It limits a user's ability and only allows them to perform a subset of system commands. Typically, a combination of some or all of the following restrictions are imposed by a restricted shell:

- Using the ?cd' command to change directories.
- Setting or unsetting certain environment variables (i.e. SHELL, PATH, etc?).
- Specifying command names that contain slashes.
- Specifying a filename containing a slash as an argument to the ?.' built-in command.
- Specifying a filename containing a slash as an argument to the ?-p' option to the ?hash' built-in command.
- Importing function definitions from the shell environment at startup.
- Parsing the value of SHELLOPTS from the shell environment at startup.

- Redirecting output using the ?>', ?>|', ?<>', ?>&', ?&>', and ?>>' redirection operators.
- Using the ?exec' built-in to replace the shell with another command.
- Adding or deleting built-in commands with the ?-f' and ?-d' options to the enable built-in.
- Using the ?enable' built-in command to enable disabled shell built-ins.
- Specifying the ?-p' option to the ?command' built-in.
- Turning off restricted mode with ?set +r' or ?set +o restricted'.

Since business needs override security a good portion of the time, it's possible that some of the above restrictions are relaxed. So do not assume that these restrictions are set in stone. I suggest you QC the quality of the restricted shell.

Reconnaissance

The first step should be to gather a little information. You'll need to know your environment. Run the ?env' command to understand how your profile is configured. You'll see which shell you're running and where your PATH is pointing to. Once you know what your PATH is, list the contents of the directory (i.e. ?ls /usr/local/rbin') to see which commands are present. It is possible you may not be able to run the ?ls' command. If not, you can use the ?echo' command with an asterisk to ?glob' directory contents if it's available:

```
echo /usr/local/rbin/*
```

You can continue on through the file system using this command to help you find other files and commands. Basically, you'll be armed with built-in shell commands as well as the ones listed in your PATH. This is your arsenal for attacking the restricted shell, but there may be exceptions as we'll find out. Once you know which commands you can execute, research each one of them to see if there are known shell escapes associated with them. Some of the techniques we're about to get into can be combined together.

Change PATH or SHELL Environment Variables

Type ?export ?p' to see the exported variables in the shell. What this will also show you is which variables are read-only. You'll note that most likely the PATH and SHELL variables are ??rx', which means you execute them, but not write to them. If they are writeable, then you can start giggling now as you'll be able to escape the restricted shell in no time! If the SHELL variable is writeable, you can simply set it to your shell of choice (i.e. sh, bash, ksh, etc?). If the PATH is writeable, then you'll be able to set it to any directory you want. I recommend setting it to one that has commands vulnerable to shell escapes.

Copying Files

If you're able to copy files into your PATH, then you'll be able to bypass the forward slash restriction. The sky is the limit at this point as you can copy commands into the PATH that have known shell escapes. You can also write your own script, copy it to the PATH, and execute it.

Another technique is to try and copy files to your home directory and execute them from there. Execution will be difficult as you will have to use ?./' in order to get it to run, and as we already know, it will fail since the restricted shell will not allow the use of a forward slash. Keep in mind, you may be able to get the commands you copy to your home directory to run if you're able to couple it with another command that has a shell escape.

Other ways you may be able to copy files or get access to them include mounting a device or file system. You may also be able to copy them to your system using a program that can copy files such as SCP or FTP.

Try to find directories other than your PATH where you can execute commands from. If you have write access to them, you can copy commands here and may be able to execute them.

Lastly, consider creating a symbolic link in a directory where you have write access and the ability to run commands

Editors

One of the most well documented techniques is to spawn a shell from within an editor such as ?vi' or ?vim'. Open any file using one of these editors and type the following and execute it from within the editor:

```
:set shell=/bin/bash
```

Next, type and execute:

```
:shell
```

Another method is to type:

```
:! /bin/bash
```

If either of these works, you will have an unrestricted shell from within the editor. Most modern restricted shells already defend against this hack, but it's always worth a shot. You may be working from a restricted editor such as rvi or rvim, which will almost certainly stop a shell from spawning. Also, try different shells with this technique and ones that follow as some restricted shells may block ?sh' or ?bash'.

Awk Command

If you can run ?awk', you can attempt to execute a shell from within it.

Type the following:

```
awk ?BEGIN {system("/bin/sh")}'
```

If successful, you'll see an unrestricted shell prompt!

Find Command

If the ?find' command is present, you can attempt to use the ?-exec' function within it.

Type the following:

```
find / -name blahblah ?exec /bin/awk ?BEGIN {system("/bin/sh")}' \;
```

Again, if successful, you'll see a blinking cursor at an unrestricted shell prompt! Note that in the above example, you are able to call the ?awk' command even if it is not present in our PATH. This is important because you are able to bypass the restriction of only being permitted to execute commands in your PATH. You are not limited to the ?awk' command.

More, Less, and Man Commands

There is a known escape within these commands. After you use the ?more', ?less', or ?man' command with a file, type ?!' followed by a command. For instance, try the following once inside the file:

```
?! /bin/sh'
```

```
?!/bin/sh
```

```
?!bash'
```

Like the shell escape in ?awk' and ?find', if successful, you'll be sitting at an unrestricted shell prompt. Note you can try different shells, and the space after the ?!' may not matter.

Tee Command

If you do not have access to an editor, and would like to create a script, you can make use of the ?tee' command. Since you cannot make use of ?>' or ?>>', the ?tee' command can help you direct your output when used in tandem with the ?echo' command. This is not a shell escape in of itself, but consider the following:

```
echo "evil script code" | tee script.sh
```

You will be able to create a file called script.sh in your home directory and add your script code to the file. Once the file is created, use the ?tee ?a' option for all subsequent commands as the ?-a' allows you to append to the file rather than overwrite the file.

Favorite Language?

Try invoking a SHELL through your favorite language:

- python: exit_code = os.system(?/bin/sh') output = os.popen(?/bin/sh').read()
- perl ?e ?exec "/bin/sh";'
- perl: exec "/bin/sh";
- ruby: exec "/bin/sh"
- lua: os.execute(?/bin/sh')
- irb(main:001:0> exec "/bin/sh"

Most likely, you will not be able to execute any of these, but it's worth a shot in case they're installed.

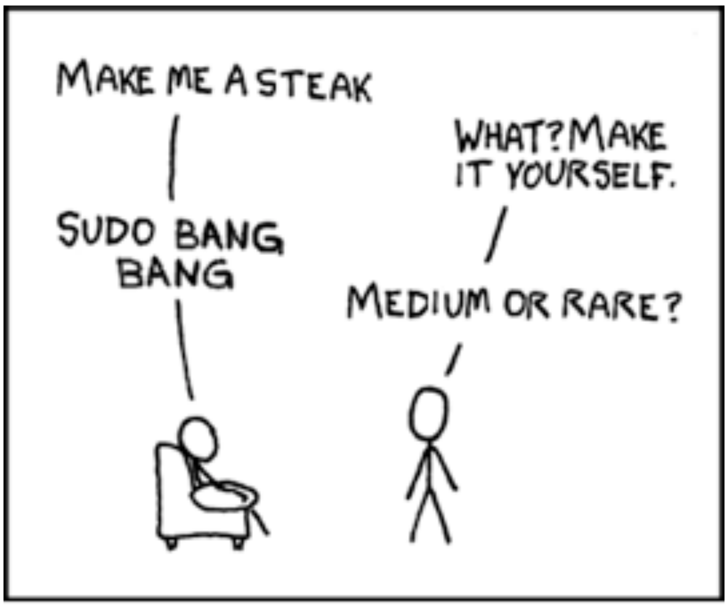
Files Executed in Unrestricted Mode?

Some restricted shells will start by running some files in an unrestricted mode before the restricted shell is applied. If your .bash_profile is executed in an unrestricted mode and it's editable, you'll be able to execute code and commands as an unrestricted user.

Conclusion

This article is far from conclusive on techniques used to escape restricted shells, but hopefully it was informative and helped you learn some new tricks. What I've come to discover is that there are many ways to attack a restricted shell. You truly are only limited by your imagination. I encourage everyone to try these methods and expand on them and develop new techniques!

Earlier, I mentioned a steak dinner was offered to me if I was able to break out of this custom restricted shell created by an engineer. I was successfully able to break out of it using multiple techniques, but when I asked him for the steak dinner, he came up with an excuse on why he wouldn't do it. I'm considering another approach:



Original comic from xkcd by Randall Munroe

Other Sources

I'd like to thank Mike Cripps and Warren Barkalow, my coworkers, for reviewing my article and providing valuable feedback. I'd also like to acknowledge and thank the folks on the GPWN list for their great input and brainstorming, especially Sterling Thomas, Jesse Bowling, and Miika Turkia.

http://www.gnu.org/s/bash/manual/html_node/The-Restricted-Shell.html

http://linuxshellaccount.blogspot.com/2008/05/restricted-accounts-and-vim-tricks-in.html

http://www.ethicalhacker.net/component/option,com_smf/Itemid,54/topic,8664.0/

Thanks for reading!

Doug Stilwell