



Add a User With Root Privileges Non-Interactively

by Pipfish
pipfish@anonymouspeach.com

My intent for this article is to provide several neat methods that can be used when working with *nix systems. I wanted to share this with folks because I think these are very useful. I'll not only tell you how to create a user whose privileges mirror root's, but I'll tell you how to do it in a non-interactive environment (via two methods). To perform these, you already need root/sudo privileges on the system in question. Of course, you must own the system or have permission to muck about with it! Doing illegal things is bad for Karma... probably.

Why?

Why would you want to add a root user if you're already root? There are probably many cases for this, but one I constantly find myself in is during penetration tests. I find myself with a non-interactive root shell on a Linux/UNIX system after taking advantage of some exploit. If I want to be able to install packages to the system (maybe a SOCKS proxy or nmap?), or do anything with much depth, I prefer an interactive environment, one where I can actually see what I'm doing and get the full benefit of TTY; namely stdin, stdout, and stderr. Some companies won't let you change root's password (or don't like it). Also, some distros don't allow the root account to log in via SSH/telnet (without changing config files). So how do I get into the system via ssh or telnet if I can't change root's password? Add a user with the same UID/GID as root, of course! Sounds easy enough, but it's tough in a non-interactive environment where any script or program that requires user input doesn't work as expected. Below we'll bypass those limitations.

Let's Do It!

The first method to add a user non-interactively is very simple. Add a user to your own system with a password and the group membership you want, then copy and echo the lines for

that user from your passwd and shadow file into /etc/passwd and /etc/shadow on the target system. I'll show you how to add a user that shares a group/userid with root in the next section, but a quick note on how: you'll want to add a user to your system with the same privileges/memberships as root.

Example: When I created a user called "test" on my system with a password of "password", this is what that user's line looked like in my passwd/shadow files:

```
my /etc/passwd:
test:x:0:0:~/home/test:/bin/sh
my /etc/shadow:
test:$6$aae8gp/j$jR0c.
HGGBDsTRRLc4
x2httg588fEj3rSjzFvZOD/nawNkPa.D
.kLzZA4UthfMc7zU8B13WuFu8oC8eK
rXxYxa/.14929:0:99999:7:::
```

On the system you have non-interactive access on, simply do this:

```
echo 'test:x:0:0:~/home/test:
/bin/sh' >> /etc/passwd
echo 'test:$6$aae8gp/j$jR0c.
HGGBDsI
RRLc4x2httg588fEj3rSjzFvZOD/nawN
kPa.D.kLzZA4UthfMc7zU8B13WuFu8
oC8eK rXxYxa/.14929:0:99999:7:::
:' >> /etc/shadow
```

The second method is a bit more involved, but can also be used/modified to script adding/changing users' passwords non-interactively. This method also demonstrates using the python crypt lib and is a good way to learn some *nix administration.

For systems that support the useradd (not adduser) command, do the following:

```
useradd username -o -u 0 -g 0
The -o switch allows multiple users to have the same uid/guid (0 is root). The user will have no password at the moment. In normal operation you'd simply issue the passwd command, but this will not work with a non-interactive shell. Assuming you have access to a system with python installed (and since the system you're
```

```
logging in from is backtrack 4 R1, I know it's got python), simply enter python and hit return.
Now you're at the >>> prompt. Type in
import crypt; print and hit enter. Next,
type crypt.crypt('<password>,<salt>'),
where password is the password you want to
assign to your user and salt is the salt value you'll
use in encryption.
The output you'll receive will be the encrypted
password. Copy it down.
Now type usermod -p encrypted
password username and hit enter. This
assigns your new user a password. Now you can
ssh in and have full interactive root access to the
system, and root's password is unchanged.
For systems that support the pw command
(FreeBSD for example), the steps are similar but
the commands are a tad different. I fooled around
a bit and found a working set of commands.
pw useradd -o -u 0 -g 0 -n username
```

The above adds the user with no password. The steps are the same for generating the encrypted password, so use python and crypt from above and copy the output.

Then enter echo encrypted_password | pw usermod -n username -h 0

The above command assigns the password to the user. Now, just as before, you have an account with root privileges, but the system's root account is unchanged.

You may ask yourself, "Why would I choose the second method rather than the first, simple echo method?" In most cases, you'll find the first method will work just fine. But the second method may be helpful if you're experimenting with scripting user add/modify actions or in some strange instance when you don't have the ability to echo commands into the passwd/shadow files.

I hope you find this useful. Good luck and happy hacking!

Simple RSA Encryption or Human-Calculable Encryption

by b3ard

At first glance, learning cryptography can be as tedious and time consuming as many other things in life, just as learning a new language can be difficult in getting familiar with the strange syntax. There are different kinds of cryptographic methods, one of them being the RSA public key cryptosystem. For a quick overview for those of you who aren't familiar with RSA, the RSA cryptosystem encrypts and decrypts messages/data by the use of public and private key pairs.

Public and private key pairs work like this: Bob and Sue have their own private and public keys. Bob and Sue both generate their own unique key pairs (using a program like the open source GnuPG), which each contain a public key and a private key. Bob doesn't know Sue's private key and vice versa; they only share their public keys. Bob uses Sue's public key to encrypt a message. Sue's public key can be received in any number

of ways, such as an online repository, in an email to Bob, or copy-pasted from Sue's website. Even a hardcopy printout of Sue's public key would suffice in a pinch. Sue receives Bob's encrypted message, and uses her private key to decrypt the message encrypted by her public key.

In practice, public and private keys are generated by using large prime numbers, and by large I mean prime numbers that are over a hundred digits long. But for quick and fast encryption when all you have is a pen, paper, and maybe a calculator, you will use extremely small prime numbers or "weak" keys to generate your cipher. For those of you who are wondering why on earth would we want to do something like this, it is because it really only works with the notion that those around us have no formal experience with cryptography, which means that there will virtually be no general or special purpose methods of attack against our cipher. So where might this be effective? The work environment where Big

