Why you should never run as root.

And what you should do when you need to.

Why not run as root?

- Bugs can damage other applications and even the system
- Security issues can be exploited by both local and remote users and used to escalate their privileges.
- Accounting (who did what when)

When do I need superuser privileges?

- Opening sockets with ports under 1024 (reserved)
- Installing things with Yum/Apt/rpm/dpkg
- Running provisioning systems (puppet/chef/salt)
- Configuring system or service settings
- Starting/Stopping services
- Restarting/Shutting down the system
- Updating the system
- Raw access to network (sniffers & packet injection)

Privilege escalation in shell

Use /etc/suders.d for custom sudo rules
Use visudo to edit and avoid getting locked out
Only use su when sudo isn't available or you don't have sudo permissions
Use setuid where applicable, be very careful with setuid/setgid around superusers.
Setuid and Setgid can only be used on binaries and not interpreted languages, but ….

Use sudo to it's full potential, don't be afraid of long sudo rules, with or without password:

%developers ALL=(packager) NOPASSWD: /bin/bash -c gpg --armor --sign -o '/tmp/sig.tar.gpg' < '/tmp/bundle*.tar'*

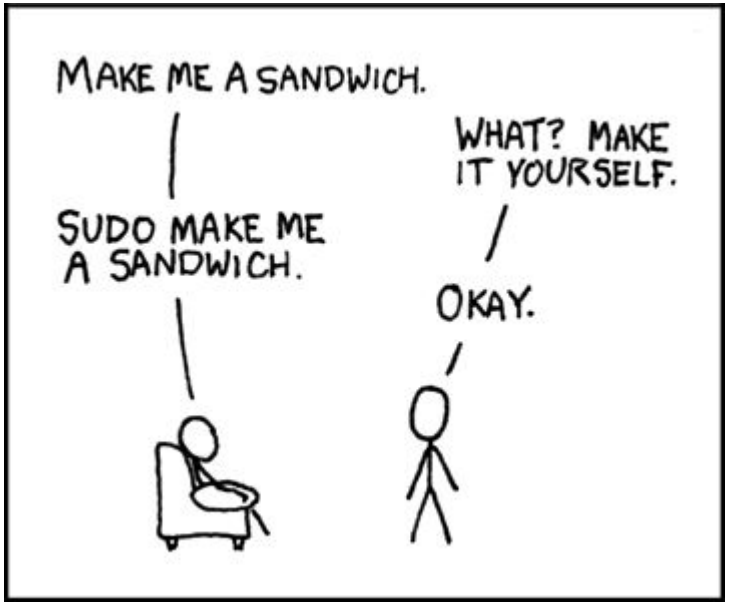
Privilege escalation (and de-escalation) in python

Python 2.6+

os.(get/set)egid
os.(get/set)gid (real)
os.(get/set)euid
os.(get/set)uid (real)
os.setregid

Python 2.7+

os.(get/set)res(uid/gid)



How do I escalate privileges inside python? ->

But this is a little boring and repetitive ……

There must be a better way right ….

```
import apt
Import os

# Open cache
cache = apt.Cache()  # NOQA

# Set effictive UID to root
ids=os.getresuid()
os.setresuid(0, ids[1], ids[2])

# Update the package list
cache.update()

# Restore original UIDs
os.setresuid(*ids)

# Re-read the cache
cache.open(None)
```

Very Much Decorators Anyone?

```
@run_as_root
@umasker
def apt_install(package_name):
    .........
```

https://goo.gl/rgeyWM

```
def run_as_root(func):
    """
    A decorator to run a code block as the root user, assumes that user has
    permissons to switch to root (see euid, ruid, suid)
    """
    def inner(*args, **kwargs):
        current_proc = multiprocessing.current_process()
        ruid, euid, suid = os.getresuid()
        rgid, egid, sgid = os.getresgid()

        # Make the actual permisson changes
        os.setresuid(0, 0, 0)
        os.setresgid(0, 0, 0)

        try:
            retval = func(*args, **kwargs)
        finally:
            # Restore original permissions
            os.setresgid(rgid, egid, sgid)
            os.setresuid(ruid, euid, suid)
        return retval
    return inner
```

https://goo.gl/j6Xonh

```
def set_uid_gid(set_rguid_to_eguid=False, set_eguid_to_rguid=False, restore_ids=True):
    """
    A decorator to set/swap real/effective UID/GID for the duration of an operation

    EUID: Effective user ID, this is what is used to check permissions
    RUID: Real user ID, this is used to determine who the original user is
          when escalating priv. to determine the original user
    SUID: Saved user ID, this is used to store original user ID when a process
          needs to temporarily de-escalate it's priv but is used to re-escalate.
    """
    def set_uid_gid_decorator(func):
        ..........................................................................
```

What about umasks?

```
def umasker(func):
    """ A decorator to change the umask while performing IO operations"""
    def inner(*args, **kwargs):
        orig_umask = os.umask(0o0002)
        try:
            retval = func(*args, **kwargs)
        finally:
            os.umask(orig_umask)
        return retval
    return inner
```

Do I have some form of superuser ID?

```
def is_superuser():
    """ Tests if a user has superuser priv."""
    if sys.version > "2.7":
        for uid in os.getresuid():
            if uid == 0:
                return True
    else:
        if os.getuid() == 0 or os.getegid() == 0:
            return True
    return False
```

But ……..

https://goo.gl/KrXQ9D

Remember that time you said “Setuid and Setgid can only be used on binaries and not interpreted languages” … ?

```
#include <Python.h>
#include <string.h>

// Accept an arbitrary number of arguments to pass on to subcommands
int main(int argc, char *argv[]) {
    PyObject *pModule, *pInst, *pHelpFunc, *pClass, *pMethod, *pFactory;
    PyObject *pArgs, *pValue;
    int i;
    ……..
```

Root is like scissors, we don’t run with it*.

*Except when we have to