

Linux Authentication and Access Control

Revised Wed. February 17, 2016

Mon, February 8 - Thu. February 18, 2016

Reading: Pollock Article, Hacking Linux Exposed



CS342 Computer Security

Department of Computer Science
Wellesley College

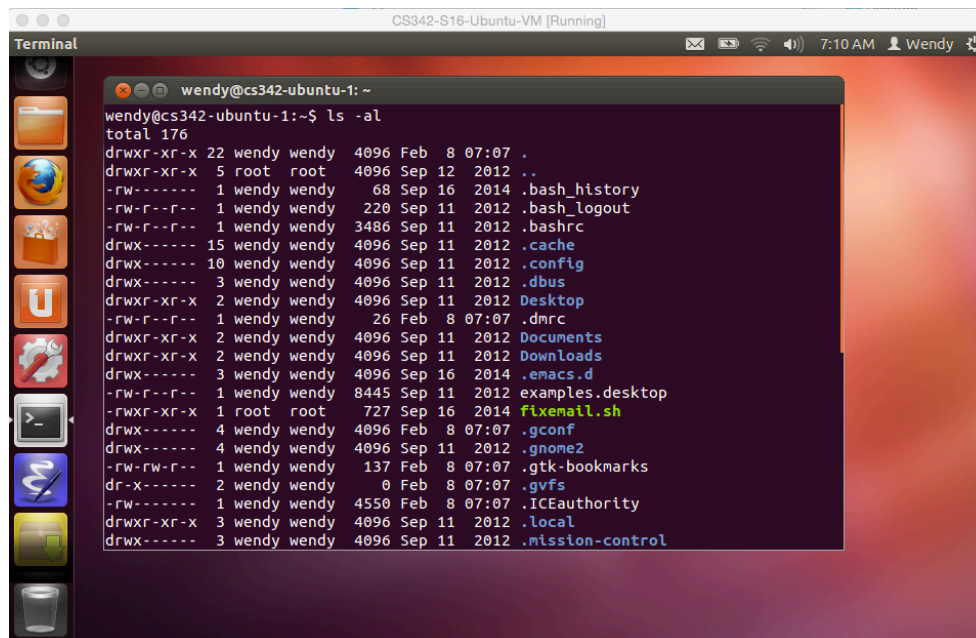
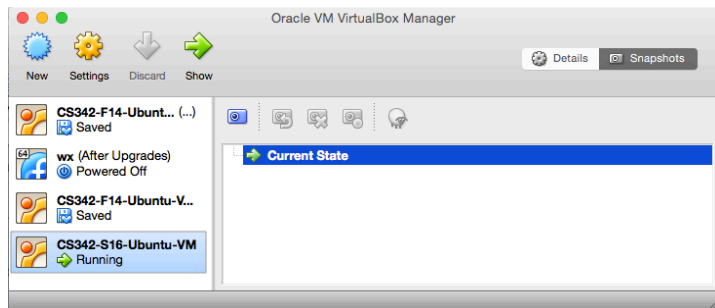
To Do Before Lab (Tue night, preferably)



1. Download and install VirtualBox 5.0.14 platform and extension package on your laptop:
<https://www.virtualbox.org/wiki/Downloads>

2. Download CS342-S16-Ubuntu-VM.ova (big file, takes a while!). Details will be posted on Tuesday.

3. Import .ova into VirtualBox as CS342-S16-Ubuntu-VM Ubuntu Linux virtual machine.



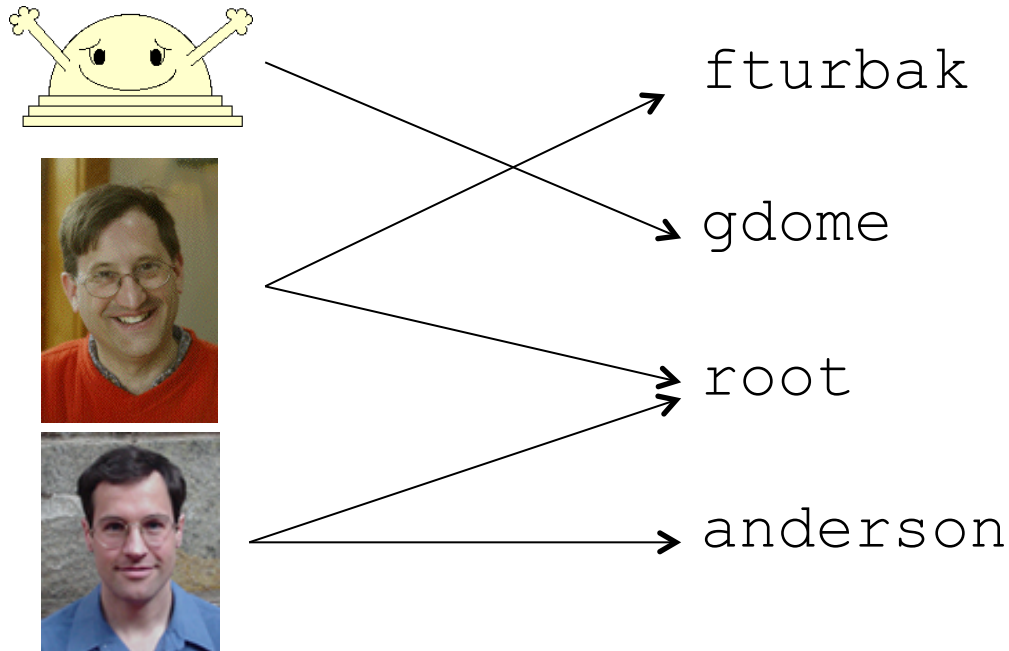
Linux Authentication and Access Control

5-2

The Big Picture: Authentication and Access Control

View system in terms of **subjects/principals**, **objects**, and **operations**:

- **Authentication** binds people/processes to subjects.



- **Access control** specifies what subjects can do with certain objects

First we'll study how Linux handles authentication and access control.
Later, we'll investigate other approaches.

Authentication: Plaintext Password File



username:	gdome
password:	albatross



albatross

```
fbar:pass34word
gdome:albatross
qsera:S&L:DSiaNW
```

password file

- Intercepted password can be entered to impersonate user.
- Password file must be private.

Hashed Password File



username:	gdome
password:	albatross



`hash(albatross) =`
`Npge08pfz4wuk`

```
fbar:Yt8w67aiewrH3
gdome:Npge08pfz4wuk
qsera:vj9awiEU9KwtF
```

password file

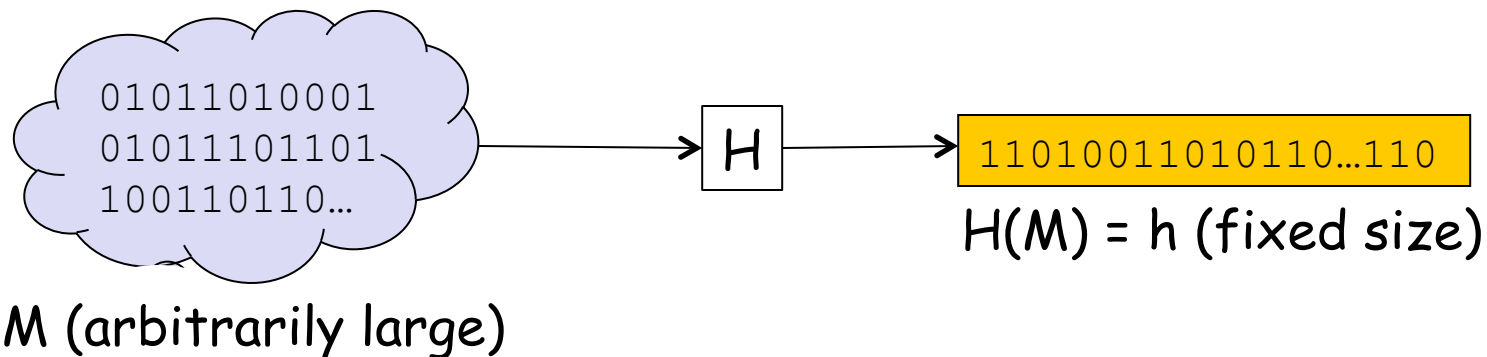
- Use one-way hash function

And now a digression on hashing ...

Digression: Cryptographic Hash Functions

A cryptographic hash function $H(M)$ returns a fixed-length hash value h for any size message M . This value h is known as a ***cryptographic checksum, fingerprint, message digest***.

They're used in passwords, digital signatures, integrity checking, etc.



Hash Function	Bits in Fingerprint
MD5 (Message Digest, Rivest, 1992)	128
SHA-1 (Secure Hash Algorithm, NSA, 1995)	160
SHA-2 (NSA, 2001)	224, 256, 384, 512
SHA-3 (NIST, 2012)	224, 256, 384, 512

Desirable Crypto Hash Function Properties

One-way (a.k.a. Pre-image Resistance):

- Easy to compute $H(M) = h$.
- Given h' , hard to find an M' such that $H(M') = h'$ (even though there are ∞ such values, by the pigeon-hole principle).
- Implies that, given M , hard to find M' such that $H(M) = H(M')$.

Collision Resistance:

- Practically difficult to find (M_1, M_2) that collide: $H(M_1) = H(M_2)$.

Seemingly Random Behavior:

- H behaves like a random mapping: changing a single bit in M should change about half the bits in $H(M)$, in an unpredictable way

Note: do not confuse cryptographic hash functions with much simpler hash functions used in hash tables!

Hashing Examples Using openssl

```
[cs342@tempest ~] echo "One if by land, two if by sea." | openssl dgst -md5  
(stdin)= fe66cbf9d929934b09cc7e8be890522e
```

```
[cs342@tempest ~] echo "One if by land, two if by sea." | openssl dgst -md5 -c  
(stdin)= fe:66:cb:f9:d9:29:93:4b:09:cc:7e:8b:e8:90:52:2e
```

```
[cs342@tempest ~] echo "One if by land, two if by sea" | openssl dgst -md5 -c  
(stdin)= 78:4b:61:4a:66:98:17:82:18:d9:25:ca:c9:64:c5:56
```

```
[cs342@tempest ~] echo "One if by land, Two if by sea." | openssl dgst -md5 -c  
(stdin)= 96:07:e8:69:cd:97:59:98:ad:21:8e:46:a8:c0:4f:0e
```

```
[cs342@tempest ~] echo "One if by land, two if by sea." | openssl dgst -sha1 -c  
(stdin)= 28:47:d1:e5:9a:96:83:bf:2f:2a:91:b8:f3:ec:21:63:d3:be:5a:6b
```

```
[cs342@tempest ~] echo "One if by land, two if by sea" | openssl dgst -sha1 -c  
(stdin)= 24:e2:d1:19:44:c2:17:49:1f:d8:9c:23:d0:9d:d2:d9:87:87:11:f1
```

More Hashing Examples

```
[cs342@tempest ~] echo "One if by land, two if by sea." | openssl dgst -sha256 -c  
(stdin)= 67:82:97:b4:e2:4f:95:28:b7:f1:cf:37:dc:8b:49:83:3f:94:d6:45:50:eb:1c:4f:79:86:  
56:0a:59:8e:e1:ed
```

```
[cs342@tempest ~] echo "One if by land, two if by sea" | openssl dgst -sha256 -c  
(stdin)= 94:09:b6:88:06:44:df:e8:47:28:e2:9c:5e:99:0c:16:76:5c:ad:1d:32:36:25:ef:2b:c3:  
0e:d8:7a:ed:38:95
```

```
[cs342@tempest ~] echo "One if by land, two if by sea." | openssl dgst -sha512 -c  
(stdin)= 93:fc:e3:a3:07:6a:90:ec:51:29:be:71:da:bf:47:dd:d0:67:ed:89:c6:b4:f9:27:ba:f6:  
c8:8c:a1:78:d2:53:ac:92:bc:3d:a5:52:06:98:d5:40:14:9f:2e:ad:fe:ab:55:ae:6f:d7:67:cf:1e:  
b4:85:0a:01:9c:78:8f:97:22
```

```
[cs342@tempest ~] echo "One if by land, two if by sea" | openssl dgst -sha512 -c  
(stdin)= 8c:89:d6:ad:9e:30:09:53:00:70:bd:a0:4c:1a:62:34:7c:5e:f2:a3:c0:25:02:99:e2:7a:  
89:b0:ac:2f:a5:fc:7c:44:72:a7:a6:69:75:45:c9:3f:18:f9:12:e0:a7:50:bf:73:c4:f8:61:42:fd:  
90:78:3d:06:28:7b:f0:48:8c
```

Hashes in Practice: Safe Downloads & Intrusion Detection

Safe downloads from untrusted sites (as long as have file hash from trusted site). Can use hashes to guarantee file integrity.

E.g., <http://www.safer-networking.org/en/download/index.html>

```
[linux@localhost ~]$ openssl dgst -md5 Desktop/spybotsd160.exe  
MD5(Desktop/spybotsd160.exe)= 0e7fbf50f87b3b7c384a2471154a7558
```

Intrusion detection: use hash (possibly combined with key) to fingerprint all important system files.

Hashed Password File (Continued)



username:	gdome
password:	albatross

hash(albatross) =
Npge08pfz4wuk

fbar:Yt8w67aiewrH3:..
gdome:Npge08pfz4wuk:..
qsera:vj9awiEU9KwtF:..

/etc/shadow password file

- Use one-way hash function.
- Intercepted hashed password cannot be entered for user.
- Password file can be public (but still better private). Early Linux had /etc/passwd public. Now separates public /etc/passwd (user info) from private /etc/shadow (hashed passwords)
- Early Linux used “crypt” function for hashing.
- Subject to password cracking

Linux IDs and /etc/passwd

Each user has a user id (uid) and belongs to (possibly several) groups each of which has a gid.

uid and default gid are stored in /etc/passwd. E.g, in my Fedora VM:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
...
lynux:x:500:500:Franklyn Turbak:/home/lynux:/bin/bash
gdome:x:501:501:Georgia Dome:/home/gdome:/bin/bash
cs342:x:502:502:CS342 Account:/home/cs342:/bin/bash
```

Password Cracking

- Online cracking (ATM PINs, keypad locks on doors, interactive password input):
 - Try common patterns followed by exhaustive brute-force search
 - Thwarted by long passwords (search space too large) and retry limitations (max of n attempts, exponential backoff).
- Offline cracking (e.g. hashed password file)
 - Examples: `john` (the Ripper), `crack`
 - Can afford more computation.
 - Step 1: plain dictionary words (including names) and user info (username).
 - Step 2: transformations: insert digits, leetspeak (1337)
 - Step 3: brute force generation
 - Thwarted by long, non-dictionary passwords that pass cracking attempts.
 - We will experiment with `john` password cracker in Lab 3 and PS2.



Thwarting Password Cracking with Salt



username:	gdome
password:	albatross

hash(d6tpFiwO,
albatross) =
↓ rBD5jd1ASAYatV2UuAlXW.

```
fbar:Yt8w67aiewrH3:...  
gdome:$1$d6tpFiwO$rBD5jd1ASAYatV2UuAlXW.:...  
qsera:$1$oVprdczq$NyoQ5WzZxeigDRBUCjQbg7:...
```

/etc/shadow password file

- “Salt” stored in password entry combined with password before hashing.
- Thwarts precomputation of large hash dictionaries.
- Linux allows /etc/shadow files with both salted MD5 passwords and old-style crypt passwords.

Hashes in Practice: Passwords in openssl

One-wayness of hashes is useful for storing passwords.

Example: suppose user gdome has password **foogle**.

- Crypt (DES) style /etc/shadow entry for gdome (first 2 chars are “salt”):

gdome:MATk8HeMV.5yk:14151:0:99999:7:::

```
[cs342@tempest ~] openssl passwd -salt Wf foogle  
rNR3vCevfquRw
```

```
[cs342@tempest ~] openssl passwd -salt Wf foogle  
Wf1x7fz3kpBrg
```

```
[cs342@tempest ~] openssl passwd -salt MA foogle  
MATk8HeMV.5yk
```

- MD5 style /etc/shadow entry for gdome (“salt” between 2nd and 3rd \$)

gdome:\$1\$TCCIG4D0\$GTUC6geaYRIq8BnhIo5n81:14151:0:99999:7:::

```
[cs342@puma ~]$ openssl passwd -1 -salt TCCIG4D0 foogle  
$1$TCCIG4D0$GTUC6geaYRIq8BnhIo5n81
```

- openssl doesn't handle SHA passwords (\$5\$ for 256; \$6\$ for 512)

Linux /etc/shadow file

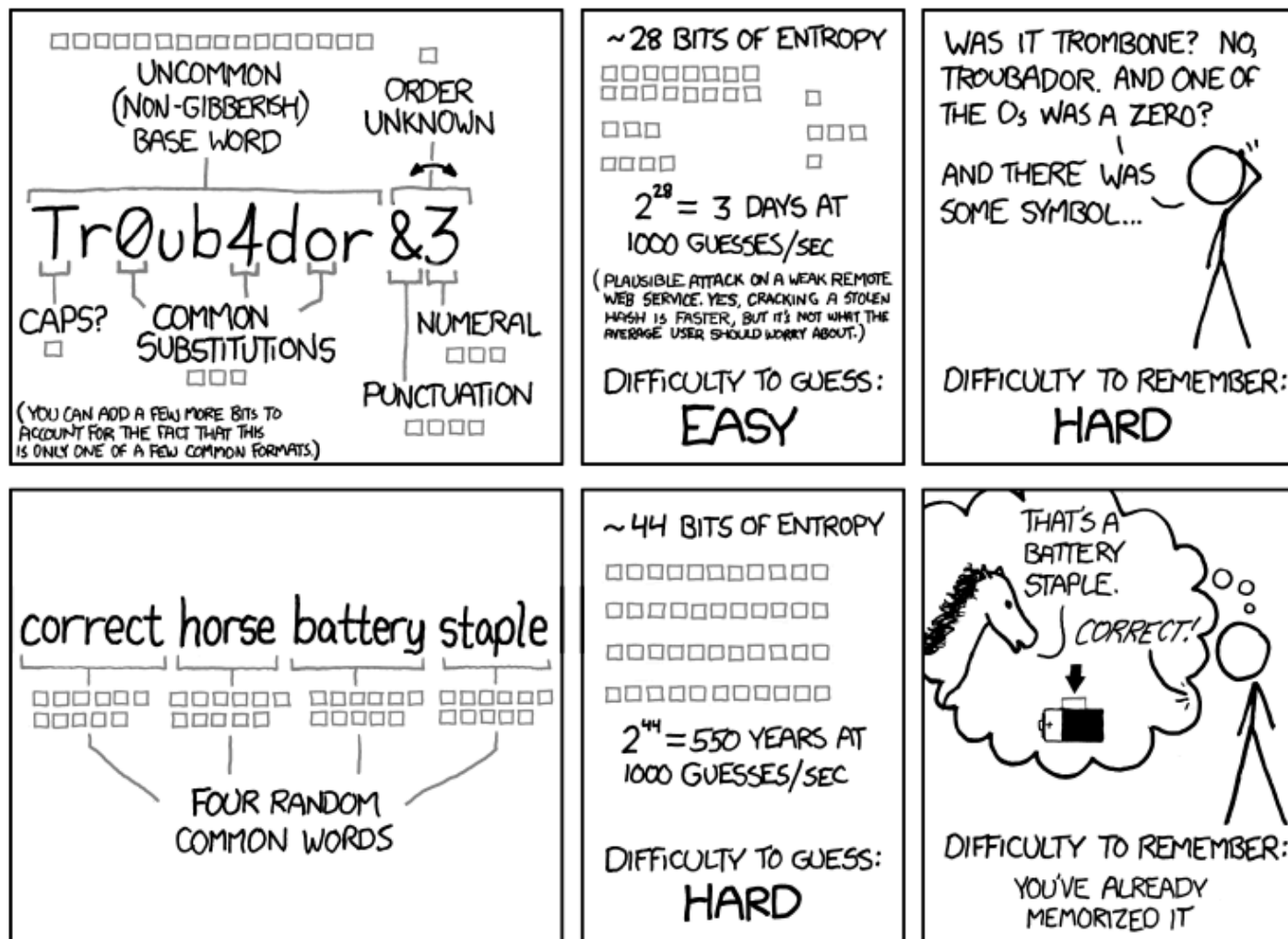
**/etc/shadow contains hashed passwords
(example from a Fedora VM, circa 2008)**

```
root:$1$SdMYD5fz$cr120C7tA0wDhXPrQHc3H1:15583:0:99999:7:::
bin:*:13665:0:99999:7:::
daemon:*:13665:0:99999:7:::
adm:*:13665:0:99999:7:::
...
lynux:$1$mUwRqvWp$2yfe5MJV0HbtdcDad8E7i/:15583:0:99999:7:::
gdome:$1$d6tpFiwO$rBD5jd1ASAYatV2UuAlXW.:14274:0:99999:7:::
cs342:$1$kmTSYJoG$0AnYYfykI2e/nWPRXdjwO.:14194:0:99999:7:::
```

**Your Ubuntu VM uses longer salts and hashes:
(this is SHA-512 salted hash for password Tr0ub4dor&3)**

```
wendy:$6$FoFIUVCi$/FUkEldzfdJXMefmv/s76m4wRpeZPnHjsVdJ9pO.QgKWuZmV
cjt5J53lZ8Sifj9Q3Pm6n6ukR9p8A1\43mnE2Q0:15594:0:99999:7:::
```

Choosing Passwords (<http://xkcd.com/936/>)



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

/etc/group

/etc/group defines groups and lists which users belong to them.

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp
mem:x:8:
kmem:x:9:
wheel:x:10:root
...
linux:x:500:
gdome:x:501:
cs342:x:502:
cs342sta:x:503:cs342,linux
cs342stu:x:504:cs342,gdome,linux
```

Sysadmins can define new groups, e.g. on puma

- faculty (all cs faculty)
- cs111sta (members of the CS111 teaching staff)
- cs251stu (cs251 students)

LDAP

Our networked dept. machines now use LDAP, a database system for users/groups that does not use `/etc/passwd` and `/etc/group` directly. (But you will be using `/etc/passwd` and `/etc/group` on your Ubuntu VM.)

On tempest, use `getent` command to get info from the LDAP database:

```
[gdome@tempest ~] getent passwd fturbak  
fturbak:x:708:708:Franklyn Turbak:/home/fturbak:/bin/bash
```

```
[gdome@tempest ~] getent passwd gdome  
gdome:x:707:707:Georgia B. Dome:/students/gdome:/bin/bash
```

```
[gdome@tempest ~] getent passwd cs342  
cs342:x:3587:501:Computer Security:/home/cs342:/bin/bash
```

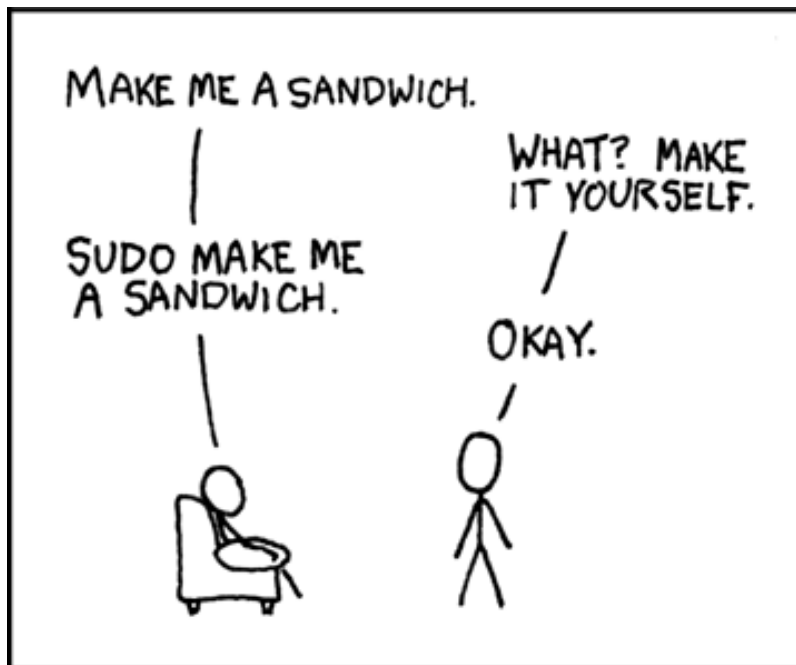
```
[gdome@tempest ~] getent group fturbak  
fturbak:x:708:
```

```
[gdome@tempest ~] getent group cs342sta  
cs342sta:x:3724:cs342,fturbak,mdawson,jakaka,cschlenk
```

```
[gdome@tempest ~] getent group faculty  
faculty:*501:anderson,ehildret,btjaden,fturbak,jherbst,pmetaxas,  
rshull,skakavou,slee,oshaer,mir,eni,susanbuck,bpw,cbassem,sravana
```

Checking/Changing Who You Are

- `whoami`: name associated with current uid
- `groups`: groups of which current uid is a member
- `su username` : "become" *username*
- `su - username`: "become" *username*, using initialization files
- `su`: "become" `root` (`su` = superuser)
- `sudo command`: Execute command as `root`
(if you're in `/etc/sudoers` and you give your password.)



<http://xkcd.com/688/>

Checking/Changing Who You Are: Examples

```
[fturbak@tempest ~] whoami
```

```
fturbak
```

```
[fturbak@tempest ~] echo $UID
```

```
708
```

```
[fturbak@tempest ~] groups
```

```
fturbak faculty ... many omitted ... cs342 cs342sta tinkerblocks
```

```
[fturbak@tempest ~] su - gdome
```

```
password: ****
```

```
[gdome@tempest ~] whoami
```

```
gdome
```

```
[gdome@tempest ~] echo $UID
```

```
707
```

```
[gdome@tempest ~] groups
```

```
gdome cs110sta cs110-staff-web ...
```

```
[gdome@tempest ~] su -
```

```
password: ****
```

```
[root@tempest ~] whoami
```

```
root
```

```
[root@tempest ~] echo $UID
```

```
0
```

```
[root@tempest ~] groups
```

```
root bin daemon sys adm disk wheel
```

```
[root@tempest ~] exit
```

```
logout
```

```
[gdome@tempest ~] exit
```

```
logout
```

```
[fturbak@tempest ~]
```

sudo in Ubuntu

The culture in Ubuntu is to use sudo rather than su to root.

- Reminds you that you're doing rooty things: less likely to accidentally do something bad (e.g. `rm -rf *` on `/`)
- Don't have to remember root password, only your own.
- Can still become root if you want to (red part not echoed)

```
wendy@cs342-ubuntu-1:~$ sudo su -  
[sudo] password for wendy: Tr0ub4dor&3  
root@cs342-ubuntu-1:~#
```

passwd: Change Password

Sudoer wendy changes guest password (red parts not echoed)

```
wendy@cs342-ubuntu-1:~$ sudo passwd guest  
[sudo] password for wendy: Tr0ub4dor&3  
Enter new UNIX password: foobar  
Retype new UNIX password: foobar  
passwd: password updated successfully
```

- As root, wouldn't need extra sudo password
- Can change anyone's password as root
- Anyone change her own password w/o being root.

useradd: Create User Account

Sudoer wendy adds user acct1

```
wendy@cs342-ubuntu-1:~$ sudo useradd -m -s /bin/bash acct1  
[sudo] password for wendy: Tr0ub4dor&3
```

- By default, puts user in empty group with same name as user
/etc/passwd now has: acct1:x:1101:1101::/home/acct1:/bin/bash
/etc/group now has: acct1:x:1101:
- Still need to set password (via passwd); ! in /etc/shadow locks user out:
acct1:!:15601:0:99999:7:::
- -m creates account /home/acct1
- -s specifies shell program for user.
- Many more options; see `man useradd`
- Add user to group sudo to make her a sudoer
- There are graphical user interfaces for creating accounts, setting passwords, etc. but you will not have access to these in many contexts (like Capture the Flag contests) So best to learn shell commands instead!

Access Control Big Picture: Subjects and Objects

Access control specifies what subjects can do with certain objects

Who are subjects?

- subject = single human (e.g., account for individual user)
- subject = multiple humans (e.g., guest account, root, groups)
- subject = system or program (e.g. apache)
- a single human may be multiple subjects (personal account, course staff, webmistress, root, ...)

What are objects?

- Data files: want to create, write, append, read, delete
- Program files: like data files, but also need to execute
- Directories: files that collect of other files and directories
- Other special files: symbolic links, devices, etc.

Access Control Matrix

		<i>Objects</i>				
		cs342 ps1	crypto program	gdome cs342 grade	fbar cs342 grade	CSDept blog
<i>Subjects/Principals</i>	gdome	r	rx	r		ra
	fbar	r			r	ra
	fturbak	rwd	rwX	rw	rw	ra
	cshanmug			rwd	rwd	r
	jstephan			r	r	r
	rpurcell					rwd

Permissions:
 r = read
 w = write
 x = execute
 a = append
 d = delete

We'll see that Linux
 provides rwX, but
 typically want more
 than that!

File Permissions in Linux

```
[lynux@localhost cs342]$ ls -al handouts
total 68
drwxrwx--- 4 lynux lynux 4096 2008-09-12 07:36 .
drwxrwxr-x 6 lynux lynux 4096 2008-09-02 03:08 ..
drwxrwx--- 2 lynux lynux 4096 2008-09-02 03:15 course-info
-rw-rw---- 1 lynux lynux 638 2008-09-09 08:59 linux-commands.txt
-rw-rw---- 1 lynux lynux 12335 2008-09-12 07:33 os-security.txt
-rw-rw---- 1 lynux lynux 3073 2008-09-11 21:27 os-security.txt~
drwxrwx--- 2 lynux lynux 4096 2008-09-09 05:38 security
```

How do you read a permission string (e.g. `drwxrwxr-x`, `-rw-rw----`)?

Leftmost chars:

- normal file
- d directory
- l link
- s socket

Other 9 chars:

- read (r), write (w), execute (x)
- permissions for 3 entities:
 - 3 chars for owner (u=user);
 - 3 for group (g);
 - 3 for everyone else (o=other)

What Do Permissions Mean?

- On file:
 - r: can read from file
 - w: can write to file
 - x: can execute file as a program
- On directory:
 - r: can list the file names in the directory.
 - w: can add new file and delete existing file
(even if don't have any permissions on file!)
 - x: can cd to directory and "search" files in directory
(i.e., get inode information necessary for file contents).
Reading/writing a regular file requires x permission
on all directories in which that file is nested!

See <http://wpollock.com/AUnix1/FilePermissions.htm> for more info!

Special permissions

There are **12** (not **9**) permission bits in Linux.

11	10	9	8	7	6	5	4	3	2	1	0
set uid	set gid	stic- key	user r	user w	user x	group r	group w	group x	other r	other w	other x

setuid bit

- on executable program: change effective user id (more later)

setgid bit

- on executable program: change effective group (more later)
- on directory: files/subdirectories inherit group and its permissions from directory

sticky bit

- on directory: only owner can delete files in the dir (used in /tmp)

Permissions as Bits & Octal Numbers

11	10	9	8	7	6	5	4	3	2	1	0
set uid	set gid	stic- key	user r	user w	user x	group r	group w	group x	other r	other w	other x

setuid bit

changes user "x" to "s"
and no "x" to "S"

setgid bit

changes group "x" to "s"
and no "x" to "S"

sticky bit

changes other "x" to "t"
and no "x" to "T"

permissions	octal
-rwxrwxrwx	777
-rwxr-x---	750
-rw-r--r--	644
-rwsr-sr--	6754
-rwsr-Sr--	6744
-rwsr-xr--	4754
-rwxr-sr--	2754
-rwxr-xr-t	1755
-rwxr-xr-T	1754

Changing File Permissions in Linux

`chmod perms file ...` : changes file permissions

`chmod -R perms file ...` : changes file permissions recursively

```
[lynux@localhost handouts]$ ls -al os-security.txt
```

```
-rw-rw---- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod o+rx os-security.txt; ls -al os-security.txt
```

```
-rw-rw-r-x 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod g-w os-security.txt; ls -al os-security.txt
```

```
-rw-r--r-x 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod u+x os-security.txt; ls -al os-security.txt
```

```
-rwxr--r-x 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod a-wx os-security.txt; ls -al os-security.txt
```

```
-r--r--r-- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod 754 os-security.txt; ls -al os-security.txt
```

```
-rwxr-xr-- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

Changing File Permissions in Linux (Continued)

```
[lynux@localhost handouts]$ chmod u+s os-security.txt; ls -al os-security.txt  
-rwsr-xr-- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod g+s os-security.txt; ls -al os-security.txt  
-rwsr-sr-- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod 754 os-security.txt; ls -al os-security.txt  
-rwxr-xr-- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ chmod 4754 os-security.txt; ls -al os-security.txt  
-rwsr-xr-- 1 lynux lynux 13290 2008-09-12 07:48 os-security.txt
```

```
[lynux@localhost handouts]$ ls -al .  
total 72  
drwxrwxr-x 4 lynux lynux 4096 2008-09-12 08:27 .  
...
```

```
[lynux@localhost handouts]$ chmod +t .; ls -al .  
total 72  
drwxrwxr-t 4 lynux lynux 4096 2008-09-12 08:27 .
```

Changing File Owner & Group: `chown` & `chgrp`

`chown username/uid file` : change owner of *file*

`chgrp groupname/gid file` : change group of *file*

`chown username/uid.groupname/gid file` : change owner & group of *file*

Notes:

- `chown` can only be executed by root.
- `chgrp` can only be executed by root or owner (and only if owner is a member of the group).
- In all cases, the `-R` flag performs recursively.

Creating a cs342stu group

To experiment with groups, let's create a cs342stu group that includes everyone but guest.

Use `sudo emacs -nw` to edit the file `/etc/group` and add a line similar to the following at the end (you may need a different group id and list of group members):

```
cs342stu:x:1103:wendy,fturbak,lynux,gdome
```

The `groups` command will list this new group in new shells, but not existing shells!

chown & chgrp Examples

```
[lynux@localhost ~]$ touch owntest; ls -al owntest  
-rw-rw-r-- 1 lynux lynux 0 2010-09-20 03:24 owntest
```

```
[lynux@localhost ~]$ chown gdome owntest  
chown: changing ownership of `owntest': Operation not permitted
```

```
[lynux@localhost ~]$ chgrp cs342stu owntest  
[lynux@localhost ~]$ ls -al owntest  
-rw-rw-r-- 1 lynux cs342stu 0 2010-09-20 03:24 owntest
```

```
[root@localhost lynux]# chown gdome.gdome owntest; ls -al owntest  
-rw-rw-r-- 1 gdome gdome 0 2010-09-20 03:24 owntest
```

```
[root@localhost lynux]# chown 502.500 owntest; ls -al owntest  
-rw-rw-r-- 1 cs342 lynux 0 2010-09-20 03:24 owntest
```

Permissions Exercise

wendy wants a `cs342test` directory containing simple text files `test1.txt` and `test2.txt`.

- Only wendy should be able to create and delete files in `cs342test`.
- Any member of the `cs342stu` group should be able to list and read the files in `cs342test`
- Anyone else (e.g. `guest`) should be able to list the file names in this directory and `cd` to the directory, but should not be able to read the files in the directory.

The SetUID (SUID) Bit

Some programs need to use protected/private files --
e.g., passwd stores encrypted passwords in /etc/shadow,
which has permissions that are something like

```
-rw----- 1 root root 1554 2008-09-15 05:57 /etc/shadow
```

(This is a white lie, but believe it for now)

How can a regular user change her own password? Because of setuid!

```
[lynux@localhost setuid]$ which passwd  
/usr/bin/passwd
```

```
[lynux@localhost setuid]$ ls -al /usr/bin/passwd  
-rwsr-xr-x 1 root root 4730 2008-09-13 08:08 /usr/bin/passwd
```

The setUID bit says that while /usr/bin/passwd is running, it will
have the owner's (in this case root's) UID as its effective UID.
So it can write to /etc/shadow!

SUID is like kernel bit in user code (see Saltzer and Schroeder paper)

Playing with SetUID

We'll now do some exercises involving setUID.

You can follow along on your Ubuntu VMs.

Download the following directory from cs.wellesley.edu:

`/home/cs342/download/setuid`

Playing with SetUID: A Squirrel program

```
/** squirrel.c */
#include <stdio.h> /* Include standard library headers */
#include <stdlib.h>

int main (int argc, char** args) {
    /* Append the argument to the file named "nest" */
    FILE *fp;
    fp=fopen("nest", "a"); /* a = append mode */
    if (fp > 0) {
        if (argc >= 1) fprintf(fp, "%s\n", args[1]);
        fclose(fp);
    } else {
        printf("Unable to open file\n");
    }
}
```

lynux's "squirrel" program appends into lynux's nest file:

```
[lynux@localhost setuid]$ touch nest; ls -al nest
-rw-rw-r-- 1 lynux lynux 0 2008-09-16 06:39 nest
```

nest is initially empty, readable by anyone, writable only by lynux

Let's Compile and Run squirrel.c

```
[lynux@localhost setuid]$ gcc -o squirrel squirrel.c ; ls -al squirrel  
-rwxrwxr-x 1 lynux lynux 5208 2008-09-16 06:45 squirrel
```

Now lynux can add items to the nest ...

```
[lynux@localhost setuid]$ cat nest  
[lynux@localhost setuid]$ ./squirrel aaa  
[lynux@localhost setuid]$ cat nest  
aaa  
[lynux@localhost setuid]$ ./squirrel bbb  
[lynux@localhost setuid]$ cat nest  
aaa  
bbb
```

... but poor gdome can't:

```
[gdome@localhost setuid]$ ./squirrel ccc  
Unable to open file
```

SUID to the Rescue

But if lynux makes the file suid ...

```
[lynux@localhost setuid]$ chmod u+s squirrel  
[lynux@localhost setuid]$ ls -al squirrel  
-rwsrwxr-x 1 lynux lynux 5208 2008-09-16 06:45 squirrel
```

... then gdome can write to it via squirrel:

```
[gdome@localhost setuid]$ ./squirrel ccc  
[gdome@localhost setuid]$ cat nest  
aaa  
bbb  
ccc
```

Can We Make Squirrel a Script?

Does linux need to write squirrel in C?

Why not just use the following bash script named "squirrel2.sh"?

```
#!/bin/bash -p
# squirrel expressed as a bash script
# The -p option says to pay attention to
#   setuid and setgid bits
if (($#>=1))
then
    echo $1 >> nest
fi
```

Problems with squirrel2 script

Lynux makes squirrel2.sh suid and takes it for a spin:

```
[lynux@localhost setuid]$ chmod u+s squirrel2.sh; ls -al squirrel2.sh  
-rwsrwxr-x 1 lynux lynux 161 2008-09-16 06:59 squirrel2.sh
```

```
[lynux@localhost setuid]$ ./squirrel2.sh ddd; cat nest  
aaa  
bbb  
ccc  
ddd
```

Sadly, gdome doesn't really execute the setuid script as lynux:

```
[gdome@localhost setuid]$ ./squirrel2.sh eee  
./squirrel2.sh: line 8: nest: Permission denied
```

Why? For safety reasons, this version of Linux does **not** allow shell scripts to be suid!

Circumventing the Restriction with `execv`

Do we have to write all `suid` programs in `C` rather than as bash scripts? Sort of ... but there's a trick to transform a bash script to a `C` program. Here's a `C` program named `squirrel3.c` that runs the script `squirrel2.sh`:

```
/* squirrel3.c */  
int main (int argc, char* argv) {  
    execv("squirrel2.sh", argv);  
}
```

Compile & Run squirrel3.c

```
[lynux@localhost setuid]$ gcc -o squirrel3 squirrel3.c
```

```
[lynux@localhost setuid]$ chmod u+s squirrel3; ls -al squirrel3  
-rwsrwxr-x 1 lynux lynux 4820 2008-09-16 07:08 squirrel3
```

```
[lynux@localhost setuid]$ echo -n "" > nest; ls -al nest  
-rw-rw-r-- 1 lynux lynux 0 2008-09-16 07:14 nest
```

```
[lynux@localhost setuid]$ ./squirrel3 eee; cat nest  
eee
```

Even gdome can use squirrel3!

```
[gdome@localhost setuid]$ ./squirrel3 fff; cat nest  
eee  
fff
```

Moral: using C's `execv`, we can execute a bash script using `suid`!

The Rootshell: A Dangerous SUID Program

```
/** rootshell.c */
#include <stdio.h>

int main (int argc, char** args) {
    /* rootshell <arg1> ... <argn>
       acts like /bin/bash -p <arg1> ... <argn> */
    char* newargs[argc + 2];
    int i;
    newargs[0] = "/bin/bash";
    newargs[1] = "-p"; /* Essential for setuid root to work */
    for (i=1; i<argc; i++) {
        newargs[i+1] = args[i];
    }
    newargs[argc + 1] = NULL; /* Array must be null-terminated */
    execv("/bin/bash", newargs);
}
```

Why rootshell shouldn't be SUID

Suppose lynux compiles it and makes it SUID:

```
[lynux@localhost setuid]$ gcc -o rootshell rootshell.c
```

```
[lynux@localhost setuid]$ chmod u+s rootshell; ls -al rootshell  
-rwsrwxr-x 1 lynux lynux 4968 2008-09-16 07:19 rootshell
```

Now what if gdome uses it?

```
[gdome@localhost setuid]$ ./rootshell
```

```
bash-3.2$ whoami
```

```
lynux
```

```
bash-3.2$ ... go off and do anything as lynux ...
```

Oops! This program allows anyone to *become* lynux in a shell!

If the owner is root, this is called a "root shell".

It's obviously very dangerous (and beloved by hackers) because it allows *Elevation of Privilege*.

Removing SUID from rootshell

This is so dangerous that we better take away SUID:

```
[lynux@localhost setuid]$ chmod u-s rootshell
```

```
[lynux@localhost setuid]$ ls -al rootshell  
-rwxrwxr-x 1 lynux lynux 4968 2008-09-16 07:19 rootshell
```

Now gdome can just create a new shell *owned by her*

```
[gdome@localhost setuid]$ ./rootshell  
[gdome@localhost setuid]$ whoami # new shell, not the original  
gdome  
[gdome@localhost setuid]$ exit # exit the new shell  
exit  
[gdome@localhost setuid]$ # now back in original shell
```

The SetGID (SGID) Bit

- On executable files, setGID bit can be used like setUID, except it changes the effective group ID of the user executing the file.
- On a directory, SGID causes new files/subdirectories to inherit permissions of the directory.

```
[lynux@localhost download]$ mkdir shared
[lynux@localhost download]$ chgrp cs342stu shared
[lynux@localhost download]$ ls -al shared
total 16
drwxrwxr-x 2 lynux cs342stu 4096 2008-09-16 08:36 .
drwxrwxr-x 4 lynux lynux    4096 2008-09-16 08:36 ..
```

```
[gdome@localhost ~]$ cd ~lynux/cs342/download/shared
[gdome@localhost shared]$ ls -al
total 16
drwxrwxr-x 2 lynux cs342stu 4096 2008-09-16 08:36 .
drwxrwxr-x 4 lynux lynux    4096 2008-09-16 08:36 ..
```

The SetGID (SGID) Bit (Continued)

```
[gdome@localhost shared]$ touch one; ls -al one
-rw-rw-r-- 1 gdome gdome          0 2008-09-16 08:43 one
```

```
[gdome@localhost shared]$ chmod g+s .
chmod: changing permissions of `.`: Operation not permitted
```

```
[lynux@localhost download]$ chmod g+s shared; ls -al shared
drwxrwsr-x 2 lynux cs342stu 4096 2008-09-16 08:43 shared
```

```
[gdome@localhost shared]$ touch two; mkdir sub; ls -al
total 32
drwxrwsr-x 3 lynux cs342stu 4096 2008-09-16 08:44 .
drwxrwxr-x 4 lynux lynux    4096 2008-09-16 08:36 ..
-rw-rw-r-- 1 gdome gdome      0 2008-09-16 08:43 one
drwxrwsr-x 2 gdome cs342stu 4096 2008-09-16 08:44 sub
-rw-rw-r-- 1 gdome cs342stu    0 2008-09-16 08:44 two
```

SGID in Practice: CS111 Drop Folders

`/home/cs111/drop:`

`drwxrwx--- 2 cs111 cs111 4096 Sep 8 11:17 .`

`lrwxrwxrwx 1 root root 29 Sep 8 11:17 astud1 -> /students/astud1/cs111/drop`

...

`/home/cs111/drop/astud1:`

`drwxr-s--- 13 astud1 cs111 4096 Sep 2 20:39 .`

`drwxr-s--- 3 astud1 cs111 4096 Sep 6 01:15 ps01`

`drwxr-s--- 3 astud1 cs111 4096 Sep 15 23:35 ps02`

...

`/home/cs111/drop/astud1/ps01:`

`drwxr-s--- 3 astud1 cs111 4096 Sep 6 01:15 .`

`drwxr-sr-x 2 astud1 cs111 4096 Sep 6 01:16 Amanda_ps01`

`/home/cs111/drop/astud1/ps01/Amanda_ps01:`

`drwxr-sr-x 2 astud1 cs111 4096 Sep 6 01:16 .`

`-rw-r--r-- 1 astud1 cs111 9044 Sep 6 01:15 Buggle.class`

...

`-rw-r--r-- 1 astud1 cs111 2678 Sep 6 01:15 Writing.java`

The Sticky Bit: Problem

Any user with write access to directory can delete a file from directory, regardless of owner:

```
[lynux@localhost shared]$ touch important; chmod 700 important
[lynux@localhost shared]$ ls -al
total 36
drwxrwsr-x 3 lynux cs342stu 4096 2008-09-16 09:06 .
drwxrwxr-x 4 lynux lynux      4096 2008-09-16 08:36 ..
-rwx----- 1 lynux cs342stu      0 2008-09-16 09:06 important
-rw-rw-r-- 1 gdome gdome        0 2008-09-16 08:43 one
drwxrwsr-x 2 gdome cs342stu 4096 2008-09-16 08:44 sub
-rw-rw-r-- 1 gdome cs342stu      0 2008-09-16 08:44 two
```

```
[gdome@localhost shared]$ rm important
rm: remove write-protected regular empty file `important'? y
```

```
[gdome@localhost shared]$ ls -al
total 32
drwxrwsr-x 3 lynux cs342stu 4096 2008-09-16 09:06 .
drwxrwxr-x 4 lynux lynux      4096 2008-09-16 08:36 ..
-rw-rw-r-- 1 gdome gdome        0 2008-09-16 08:43 one
drwxrwsr-x 2 gdome cs342stu 4096 2008-09-16 08:44 sub
-rw-rw-r-- 1 gdome cs342stu      0 2008-09-16 08:44 two
```

The Sticky Bit: Solution

The "sticky bit" on a directory allows users to delete only those files owned by them. Used to protect files in shared directory /tmp

```
[lynux@localhost shared]$ chmod +t .  
[lynux@localhost shared]$ touch important2  
[lynux@localhost shared]$ chmod 700 important2; ls -al  
total 36  
drwxrwsr-t 3 lynux cs342stu 4096 2008-09-16 09:09 .  
drwxrwxr-x 4 lynux lynux      4096 2008-09-16 08:36 ..  
-rwx----- 1 lynux cs342stu      0 2008-09-16 09:09 important2  
-rw-rw-r-- 1 gdome gdome        0 2008-09-16 08:43 one  
drwxrwsr-x 2 gdome cs342stu 4096 2008-09-16 08:44 sub  
-rw-rw-r-- 1 gdome cs342stu      0 2008-09-16 08:44 two
```

```
[gdome@localhost shared]$ rm important2  
rm: remove write-protected regular empty file `important2'? y  
rm: cannot remove `important2': Operation not permitted
```

Password-protecting Web Pages: .htaccess

Can require an HTTP password on a directory via .htaccess file in it.

```
# begin file .htaccess
AuthUserFile /var/www/htpasswd
AuthGroupFile /dev/null
AuthName ByPassword
AuthType Basic

<Limit GET>
    require user gdome # prompt gdome for http password
</Limit>
# end file .htaccess
```

Some other options are helpful

```
allow from .wellesley.edu # allow those from wellesley domain
allow from 149.130. # allow those from wellesley machines
require valid-user # allow any user with http password,
                  # not just gdome
```

Setting an HTTP password

Can set HTTP password for gdome by executing the following as root:

```
# The first time (-c creates password file; -m uses MD5 hashing)  
htpasswd -c -m /var/www/htpasswd gdome
```

```
# Subsequent times  
htpasswd -m /var/www/htpasswd gdome
```

Warning: http passwords are sent in the clear!
(We'll see this later in semester using Wireshark.)

Real-Life Access-Control Design Problem

Goal: CS111 pset solutions viewable on web by staff but not students.

Important details:

- don't want solutions to be world-readable on tempest
- viewable web pages must be readable by user apache.

How to achieve this goal?

On PS3, you'll be given a challenging related permissions problem!

File Attributes

In addition to permissions, Linux ext2 files also have **attributes** settable via `chattr` (by root only) and gettable via `lsattr`.

These include:

- **a** : (append only) Can only append to file with this attribute.
- **c** : (compressed) A file with this attribute is compressed when written to disk and uncompressed when read from disk.
- **i** : (immutable) A file with this cannot be modified, deleted or renamed, and no link can be created to this file.

See man pages and *Hacking Linux Exposed* for more details.

File Attribute Example: i flag

```
[lynux@localhost shared]$ touch itest; ls -al itest  
-rw-rw-r-- 1 lynux cs342stu  6 2010-09-20 03:01 itest
```

```
[lynux@localhost shared]$ echo "foo" > itest; cat itest  
foo
```

```
[root@localhost shared]# chattr +i itest  
[root@localhost shared]# lsattr itest  
----i----- itest
```

```
[lynux@localhost shared]$ echo "bar" > itest  
bash: itest: Permission denied
```

File Attribute Example: a flag

```
[lynux@localhost shared]$ touch atest; ls -al atest  
-rw-rw-r-- 1 lynux cs342stu  6 2010-09-20 02:58 atest
```

```
[lynux@localhost shared]$ echo "foo" > atest; cat atest  
foo
```

```
[root@localhost shared]# chattr +a atest  
[root@localhost shared]# lsattr atest  
-----a----- atest
```

```
[lynux@localhost shared]$ echo "bar" > atest  
bash: atest: Operation not permitted  
[lynux@localhost shared]$ echo "bar" >> atest; cat atest  
foo  
bar
```