

אבטחת מחשבים ורשתות תקשורת

(372-1-4601)

מטלה מס' 2

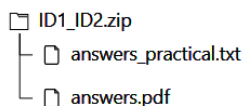
מתרגל אחראי: חן דויטשמן

הנחיות כלליות:

1. המטלה מורכבת משני חלקים – חלק תיאוראי וחלק מעשי.
2. שאלות על המטלה יש לשאול אך ורק בפורום הייעודי ב-Moodle. שאלות אשר ישאלו בדוא"ל לא יענו.
3. על התשובות להיות קצרות אך מלאות.
4. כחלק מתהליך בדיקת העבודה תתבצע בדיקה לזיהוי עבודות מועתקות. כל מקרה של העתקה יטופל בחומרה ע"י ועדת משמעת אוניברסיטאית.

הנחיות להגשה:

1. ההגשה היא בזוגות בלבד.
2. על התשובות של החלק התיאורטי להיות מוקלדות במחשב או סרוקות בכתב יד נקי וברור.
3. יש לענות על השאלות ולהפיק קובץ pdf עם התשובות שלכם.
4. לאחר סיום החלק המעשי, עליכם להריץ סקריפט (מצורף ל-zip) שמתאר את המערכת שלכם לאחר השינויים שביצעתם בה. קובץ הטקסט שהסקריפט מפיק מהווה את ההגשה שלכם לחלק המעשי.
5. את שני הקבצים (קובץ pdf לחלק התיאורטי, קובץ txt לחלק המעשי) יש לכווץ בקובץ zip ולהגיש בתיבת ההגשה ב-Moodle.
6. שם הקובץ יהיה ID1_ID2.zip כאשר ID1 ו-ID2 מוחלפים בת.ז של מגישי המטלה.
7. כלומר, מבנה ההגשה יהיה כלהלן:



8. רק אחד מחברי הצוות יגיש את המטלה ב-Moodle.
9. כל איחור של יום בהגשה יגרע 5 נקודות מהציון.

בהצלחה

חלק תיאורטי (60 נק')

שאלה מס' 1 – פרוטוקולי אימות א' (10 נק')

נתון פרוטוקול המאפשר לשני לקוחות A ו- B לייצר מפתח סימטרי משותף K_{AB} בעזרת שרת אמון S .

$$A \rightarrow B : A$$

$$B \rightarrow A : N_B$$

$$A \rightarrow B : \{N_B, K_{AB}\}_{K_{AS}}$$

$$B \rightarrow S : \{A, B, \{N_B, K_{AB}\}_{K_{AS}}\}_{K_{BS}}$$

$$S \rightarrow B : \{N_B, K_{AB}\}_{K_{BS}}$$

הפרוטוקול איננו מאובטח.

1. (3 נק') תאר את ההתקפה והדגם את תרחיש התקיפה.
2. (3 נק') מהן היכולות הנדרשות מן התוקף בכדי לממש את ההתקפה?
3. (4 נק') כיצד ניתן לתקן את הפרוטוקול בכדי להתמודד עם ההתקפה שתיארת?

שאלה מס' 2 – פרוטוקולי אימות ב' (10 נק')

שאלה זו עוסקת בפרוטוקול ה- $handshake$ של TLS . TLS הוא גירסה מעודכנת של SSL , אחד מפרוטוקולי האבטחה הפופולריים והחשובים ביותר של רשת האינטרנט. שימו לב: בפתרון שאלה זו אין חשיבות להיכרות מוקדמת עם שלבי הפרוטוקול. התמקדו בשלבים העיקריים ולא בפרטים הטכניים.

להלן השלבים העיקריים של הפרוטוקול:

1. הלקוח שולח הודעת $ClientHello$ לשרת.
2. השרת שולח $ServerHello, Cert_s$ ללקוח.
3. הלקוח מוודא את הסרטיפיקט $Cert_s$ שהתקבל מהשרת ומוציא ממנו את המפתח הפומבי של השרת PK_s . לאחר מכן הלקוח בוחר מחרוזת $PreMasterKey$ באקראי, מצפין אותה עם המפתח הפומבי של השרת ושולח לשרת את $Enc_{PK_s}(PreMasterKey)$.
4. השרת מפענח את המחרוזת באמצעות המפתח הפרטי שלו ומקבל את ה- $PreMasterKey$. השרת שולח ללקוח מחרוזת $ServerFinished$ שהיא פונקציה של ה- $PreMasterKey$ וערכים קודמים שנשלחו במהלך הפרוטוקול.
5. הלקוח מסיים את אימות השרת ע"י זה שהוא מחשב בעצמו את ה- $ServerFinished$ ומשווה לזה שהתקבל מהשרת.
6. בסופו של הפרוטוקול, ללקוח ולשרת יש זוג מפתחות סימטריים משותפים K_{MAC}, K_{Enc} להצפנה ואימות שמחושבים ע"י הפעלת פונקציית $hash$ מסוימת H :

$$K_{MAC}, K_{Enc} = H(PreMasterKey, transcript)$$

כאשר $transcript$ כולל את ההודעות הגלויות שנשלחו ע"י הצדדים במהלך הפרוטוקול.

K_{MAC}, K_{Enc} משמשים ליצירת ערוץ תקשורת בטוח בין הלקוח לשרת.

על מנת "לחזק" את הפרוטוקול, הוצע להוסיף את השלבים הבאים בסוף הפרוטוקול שמאמתים את השרת פעם נוספת:

7. הלקוח בוחר מחרוזת r באקראי, מצפין אותה עם המפתח הפומבי של השרת ושולח לשרת את $Enc_{PK_S}(r)$.
8. השרת מפענח את המחרוזת באמצעות המפתח הפרטי שלו, מקבל את r ושולח ללקוח.
9. הלקוח מוודא שקיבל את המחרוזת r ששלח.

כל התקשורת בשלבים 7,8,9 מוצפנת ומאומתת באמצעות המפתחות הסימטריים K_{MAC}, K_{Enc} .

הראו שהשינוי הנ"ל אינו בטוח ע"י תיאור התקפה בה תוקף מאזין לפרוטוקול ה-*handshake* בין לקוח לבין *amazon.com* ולאחר מכן יכול לחשב את המפתחות הסימטריים K_{MAC}, K_{Enc} עליהם הוסכם (וכך לפענח את כל התקשורת בין הלקוח ל-*amazon.com* או לשנות הודעות כרצונו).

הדרכה: התוקף יכול לתקשר בעצמו עם *amazon.com*.

שאלה מס' 3 – פרוטקולי אימות ג' (15 נק')

הפרוטוקול הבא משתמש באלגוריתם **RSA** לצורך שליחת הודעה (**M**) מאליס (**A**) לבוב (**B**) באופן מאובטח.

1. **A** שולח ל-**B** בקשה לסרטיפיקט
2. **B** שולח ל-**A** את הסרטיפיקט שלו – $Cert_B$
3. **A** בודק את הסרטיפיקט
4. **A** שולח הודעה **M** ל-**B** מוצפנת עם המפתח הפומבי של **B**

ניתן להניח כי:

- לכל משתמש יש מפתח פומבי, מפתח פרטי וסרטיפיקט
- במערכת קיים גורם אמון (CA) אשר כולם מכירים את המפתח הפומבי שלו
- המשתמשים מכירים את ה-**ID** של האחר

נגדיר את הסימון הבא:

סימון	הגדרה
PUB_x	המפתח הפומבי של X
PR_x	המפתח הפרטי של X
$E_k(M)$	הצפנת RSA של ההודעה M עם המפתח K

סטודנטים בקורס אבטחת מידע הציעו מספר דרכים למימוש הסרטיפיקט. עבור כל אחת מהדרכים:

1. (3 נקודות) תאר במדויק את הפעולות אותן צריכה לבצע **A** בכדי לבדוק את הסרטיפיקט.
2. (12 נקודות) ציין האם הפרוטוקול בטוח, כלומר האם ייתכן שמשתמש אחר (לדוגמא **C**) יוכל להתערב בתקשורת בין **A** ל-**B** ובעזרת כך לגלות את תוכן ההודעה **M**. במידה ותשובתך היא "הפרוטוקול בטוח" הסבר מדוע. במידה ותשובתך היא "הפרוטוקול אינו בטוח" עליך לתאר ולהדגים באמצעות תרשים התקפה על הפרוטוקול אותה יכול לבצע **C** על מנת לגלות את תוכן ההודעה (תשובות שלא יכללו תיאור מפורט ותרשים לא יקבלו ניקוד).

להלן הצעות הסטודנטים:

- א. $Cert_x = E_{PRCA}(PUB_x, ID_x)$
 ב. $Cert_x = (E_{PRCA}(PUB_x), E_{PRCA}(ID_x))$
 ג. $Cert_x = (PUB_x, E_{PR_x}(E_{PRCA}(ID_x)))$
 ד. $Cert_x = (PUB_x, E_{PRCA}(E_{PR_x}(ID_x)))$

שאלה מס' 4 – POSIX Access Control 'א' (10 נק')

לאחר הרצת הפקודה `ls -l` במערכת מבוססת Linux התקבל הפלט הבא:

-r-xr-sr-x	1	charlie	acct	70483	2008-01-04	22:53	accounting
-r--rw----	1	alice	acct	139008	2008-05-13	14:53	accounts
-rwxr-xr-x	1	system	system	230482	1997-04-27	22:53	chmod
-rw-r--r--	1	alice	users	7072	2008-06-01	22:53	cv.txt
-r--r-----	1	bob	gurus	19341	2008-06-03	13:29	exam
-r--r-----	1	alice	gurus	6316	2008-06-03	16:25	solutions

- המשתמשים alice ו-bob חברים בקבוצה users.
- השתמש charlie חבר בקבוצה gurus.
- האפליקציה chmod הינה כפי שמתוארת ב-Unix Manual (משתמשים יכולים להריץ את האפליקציה במידה ויש להם הרשאות לכך).
- האפליקציה accounting מאפשר לצרף רשומה (append) לקובץ ה-accounts.

מלא/י את טבלת ההרשאות הבאה עבור המשתמשים alice, bob ו-charlie והקבצים accounts, cv.txt, exam, solutions אשר מראה עבור כל משתמש האם הוא יכול לבצע קריאה (Read), כתיבה (Write) או רק צירוף של רשומה (Append) עבור על אחד מהקבצים.

	accounts			cv.txt			exam			solutions		
	R	W	A	R	W	A	R	W	A	R	W	A
alice												
bob												
charlie												

- לא קיים שיתוף פעולה בין משתמשים במערכת.
- הפעולה Append עומדת בפני עצמה והיא איננה זהה לכתיבה (Write) לסוף הקובץ.

שאלה מס' 5 – POSIX Access Control ב' (15 נק')

לפניך תצורת ה-DAC (discretionary access-control) במערכת Unix בארגון כלשהו:

alex, benn ו-cloe חברים בקבוצה staff.

Cloe חברה בקבוצה gurus.

הקובץ microedit הוא אפליקציה שהיא העתק של האפליקציה /usr/bin/vim שהיא עורך טקסט רגיל שניתן להריץ.

```
$ ls -ld . * */*
drwxr-xr-x 1 alex staff 32768 Apr 2 2010 .
-rw----r-- 1 alex gurus 31359 Jul 24 2011 manual.txt
-r--rw--w- 1 benn gurus 4359 Jul 24 2011 report.txt
-rwsr--r-x 1 benn gurus 141359 Jun 1 2013 microedit
dr--r-xr-x 1 benn staff 32768 Jul 23 2011 src
-rw-r--r-- 1 benn staff 81359 Feb 28 2012 src/code.c
-r--rw---- 1 cloe gurus 959 Jan 23 2012 src/code.h
```

ציירי/ את ה-access control matrix שמראה לכל קובץ מחמשת הקבצים שלעיל (מהפלט של ls -ld), האם alex, benn או cloe יכולים לקבל הרשאות קריאה, כתיבה או הרצה לקבצים.

הדרכה: חשבו את ההרשאות המקסימליות אותן יכול ה-user להשיג.

	manual.txt	report.txt	microedit	src/code.c	src/code.h
alex					
benn					
cloe					

חלק מעשי (40 נק')

POSIX File Permissions

Setup

The practical part of this assignment will be conducted using a virtual machine of Ubuntu Server 14.04.

- (1) Use the tutorial on Moodle for a step-by-step instructions on how to install the VM and log into it using SSH.*
- (2) When running the script to extract your answers (submit.sh) please redirect the output to a file (.txt) and attach it to the submission.*

Now you can start working on the task 😊.

Overview

The purpose of this exercise is to introduce you to filesystem and network access control schemes and the "principle of least privilege" using POSIX filesystem permissions.

After this exercise, you will:

- Understand the POSIX permissions structure including SUID and SGID bits.
- Understand the essence of the **sudo** utility and how to configure and use it securely.
- Be able to apply that knowledge to configure permissions in multiple scenarios, such as:
 - shared system directories
 - user home directories and private directories
 - privileged system directories
 - unprivileged temporary directories
 - editing important configuration files
 - restarting system processes.
 - potential privilege escalation problems

Practical Assignment


You are interviewing to the system administrator role in a company. You will be presented with some tasks and questions. Your entire work should be done only on the `server` machine.

The following sections includes several permissions and file creation exercises. You are fully encouraged to use the Internet, man pages, help screens, and any other resources available to you in the execution and answering of these problems.


Please read and use the disambiguation rules (later in this document) for setting the correct permissions. Also, make sure that you test your answers - POSIX permissions are simple in theory, but in practice many combinations have counterintuitive effects!

Your answers will consist of a text file, created by the `submit.sh` script described above. Please redirect the script output (echo) to a `.txt` file (i.e. `./submit.sh > answers_practical.txt`) and pull it from the VM to your local machine.

Home Directory Security (20 points)

 **Note:** Admins - members of the `wheel` group have `sudo` access. However, unless instructed to do so, use only standard UNIX ACLs - don't give user accounts `sudo` permissions or consider the `sudo` access the 'wheel' group already has. Of course, *you* need to use `sudo` to create the accounts, edit root files, etc... this is exactly what `sudo` is for.

Your server needs two home directories, the usual `/home` and also `/admins` for your team. Normal home directories are private, while the home directories in `/admins` will be publicly viewable and somewhat collaborative.

 You can test the permissions settings on `server` by logging in as the various accounts you've created! For example, after creating the account `larry`, you can log in by executing `ssh larry@localhost` and entering the password you used for `larry`. In this way, you can see if the permissions you set meet the requirements.

1. Create the `/admins` directory.
2. Create the groups `emp` and `wheel`
3. Create the user accounts `larry`, `moe`, and `curly`. You may set the passwords to anything you like.
4. Add the three accounts you just made to the `emp` group. In our system, accounts in the `emp` group are "normal" (i.e., non-admin) accounts.
5. Next, we will set up our administrators. Create the user accounts `ken`, `dmr`, and `bwk` - specifying that the home directory for each admin should be located at `/admins/username` - where `username` is `ken`, `dmr` or `bwk`. In other words, admin homedirs are **not** in `/home`. You may set the passwords to anything you like.

6. Add the admin accounts to the `wheel` group (Ensure that admins are **not** part of the `emp` group.) Being in the `wheel` group is what gives these users administrator rights (i.e., `sudo` powers).
7. On this system, default permissions for new home directories allow `other` users (i.e., users that are not the `owner` or in the specified `group`) to read and execute files in the directory. This is too permissive for us. Set the mode on the home directories in `/home` so that `owner` can read, write, and execute, `group` can read and execute and `other` has no permissions. (Set the mode on the `homedir` only -- do not set it recursively.)
8. Individual home directories should now be inaccessible to `other` users. Now, set the permission mode on `/home` itself so that normal users can't list the contents of `/home` but can still access their home directories *and* so that members of the `wheel` group have full access to the directory (without using `sudo`).
9. By default, each `homedir` is *owned* by its user, and the `homedir`'s *group* is set to the group named after the user. (For example, `ken`'s `homedir` is set to `ken:wheel` - i.e., `ken` is the `owner` and the `group` is set to `ken`'s `group`.) Set the permission *modes* recursively on the individual home directories in `/admins` (see `man chmod`) so that:
 - o `owners` have *full access*
 - o `group` users (users who are in the `group` associated with a user's home directory) can read and create files in that `homedir`
 - o `other` users can read and execute any files (unlike the home directories in `/home`)
 - o files created by a `group` member in that `homedir` should be set to the `homedir` `owner`'s `group`. (Hint: Look up what the `SUID` and `SGID` bits do on directories.)

The Ballot Box (7 points)

All regular employees use this directory to submit votes for "employee of the month".

1. Create the `/ballots` directory.
2. Set the permissions on `/ballots` so that it is owned by `root` and users can write files into the directory but cannot list the contents of the directory.
3. Furthermore, set it so that members of the `wheel` group have no access (not including `sudo`).

The TPS Reports Directory (8 points)

Admin employees submit TPS reports in this partially collaborative directory.

1. Create the `/tpsreports` directory.
2. Create the `tps` user.
3. Set the permissions on `/tpsreports` so that it is owned by `tps` and that `tps` and members of the `wheel` group have full access to the directory, but so that no one else has access to the directory.

4. Furthermore, configure it so that files written into it are still owned by the `wheel` group, but so that one member of `wheel` cannot delete another member's files.

Editing Configuration Files (5 points)

All members of the `wheel` group do system administration on the server. Because of this, they all have full `sudo` access to `root` privilege with the `/etc/sudoers` directive `'%wheel ALL=(ALL) NOPASSWD: ALL'`.

The "NOPASSWD" means they don't have to enter a `passwd` upon `sudo` invocation.

1. Add the above directive to `/etc/sudoers` to allow users from `wheel` group to perform system administration (i.e., run `sudo`). You can use the `vim` text editor to do so.

Done!

Rules for resolving filesystem permission ambiguities:

Permissions should *always* be set to reflect the **least privilege** required to fulfill the requirements. In POSIX permissions, every bit set represents *less security*, so we want to set as few bits as possible. Resolve any ambiguities with this cumulative list of guidelines:

1. Files you are instructed to create are to be owned by `root` unless otherwise specified. Doing the work as `root` should do this automatically. (You can become root by executing `sudo su -.`)
 - Exception: Files in homedirs should be owned by the home directory owner (`useradd` should do this by default for boilerplate files like `.bash_login`, etc).
2. When setting ownership of a file, set the group class to the same thing as the user (owner) class unless otherwise specified.
3. Files whose permissions you are *not* otherwise instructed to change should stay at their default.
4. For any files, whose permissions you *are* instructed to change, unspecified permissions are **always** assumed to be 0 (no access). In other words, instructions for setting file permissions implicitly include all access groups (even those not explicitly mentioned).
5. Permissions are assumed to be for all classes unless specified.
6. When granting permissions, `setuid`, `setgid`, and sticky bits are **never** granted unless specifically required to solve the problem. These bits are special and must be required by the nature of the question or be otherwise mentioned to be granted.
7. If any tasks are not possible with the standard POSIX permissions available in this exercise, explain why.

Recommended Reading

This section contains information that will help you through the exercises in this assignment, though not mandatory, reading this section is recommended!

The *sudo* command

Some applications simply require stronger guarantees, finer granularity, or other features that the traditional permissions cannot express. For those applications, users have many alternatives. `sudo` is one simple extension to traditional Unix permissions that has become very popular.

sudo is a setuid root application with its own ACL (stored in `/etc/sudoers`) that specifies, with fine granularity, tasks that users and groups can perform as root. For example, `sudo` could be used to allow a user to act as root in order to kill processes with a specific name. The user would otherwise have no additional privileges. This is an example of a perfect use for setuid root programs -- granting strongly-constrained privileges to unprivileged users.

However, `sudo` is a double-edged sword. On the one hand, it greatly enhances the expressiveness of Unix permissions without actually changing the permissions system. On the other hand, if improperly configured, it offers **easy access to root**. For more information on `sudo` and the `sudoers` ACL format, please see the manpages for `sudo` and `sudoers` (the configuration file), or other `sudo`-related material online (in particular regarding `sudo` exploits).

Beyond `sudo`, two broader and more revolutionary alternatives for Linux permissions include SELinux and grsecurity, while other options exist including LDAP, Novell eDirectory, and other permissions systems for other operating systems. (We won't be using any of these.)

Software Tools (We discussed most of them on PS6)

Add users to a system (related tools – `adduser`, `chfn`, `passwd`):

`adduser` is the tool available for adding users to the system. To create a new user, execute:

```
$ adduser username
```

`adduser` will copy files from the `/etc/skel` directory to become the new homedir of the new user in the `/home/` directory. You can specify a different home directory or automatically add the new user to groups; those options can be found in the `adduser` manpage.

`adduser` does not set any `finger` information for the user (this is not strictly necessary anyway), but the tool `chfn` will do that:

```
$ sudo chfn jimbo
Changing finger information for jimbo.
Name []: ...
```

By default, the user is created with a "locked out" account with no password set. To set the password, use the command `passwd`:

```
$ sudo passwd jimbo
Changing password for user jimbo.
New Unix password:
Retype new Unix password:
passwd: all authentication tokens updated successfully.
```

`passwd` will complain if it doesn't like the password you enter, but will accept anything with enough prodding.

Once an account is created, you can log into it in a number of ways:

1. If logged in to the system where you created the account, you can execute `ssh newuser@localhost` to reconnect locally as the new user with the new password.
2. If local, you can also use the `su` command `su newuser` to change to that user. This does not always update all access credentials, however. This method can also be used to become root by entering `sudo su -`.
3. If logged in to `users.create.iucc.ac.il`, you can `ssh` to the node as the new user.

Add groups to a system(related tool – `groupadd`)

`groupadd` adds a new group to the system with a unique ID (by default). Example:

```
$ groupadd newgroup
```

See `man groupadd` for more information.

Modify a user (related tool – `usermod`)

`usermod` can be used to modify many details of a preexisting user account. For more information, see `man usermod`.

Change ownership of a file (related tools – `chown`, `chgrp`)

`chown` stands for **change ownership** and is (unsurprisingly) used to change the owner and group of a file.

The syntax is very simple. To change the owner of a file, execute:

```
$ chown newowner filename
```

To change the owner and group classes of a file, execute:

```
$ chown newowner:newgroup filename
```

`chgrp` stands for **change group** and works very similarly to `chown`. To change the group of a file, execute:

```
$ chgrp newgroup filename
```

Recursive and other options exist; see `man chown` or `man chgrp` for more information.

Change the mode of a file (related tool – `chmod`):

`chmod` stands for **change mode** and is used to change the permissions mode of a file. Earlier, we discussed how the POSIX ACL has three access classes. User (or owner), group, and other (or world). The permissions mode for each access class can be changed by the `chmod` command.

There are two ways to use `chmod`; one is an *absolute numeric mode* and the other is a *symbolic mode*.

Absolute -- setting the permissions explicitly

In the absolute mode, `chmod` takes a file mode in 3 or 4 digits, each of which represents the absolute permission mode for one access class expressed in octal, where each number is a sum of the permission bits. Each permission has a unique value: `read` permission is 4, `write` permission is 2, and `execute` permission is 1. These values represent the position of the permission in a 3-bit value.

For example, the mode `777` means "full permissions" because all bits are set in each access class. Similarly, `000` means "no permissions" because all bits are unset in each access class.

The 3-digit mode `777` is the equivalent to the 4-digit mode `0777` where the leading `0` represents the "special" permission class of `setuid`, `setgid`, and `sticky`. Likewise, the modes `000` and `0000` are equivalent and represent the absence of permission. (The owner of the file and root still have the ability to change the file's mode by virtue of their ownership and

superuser status.) Finally, if the 3-digit mode is used, `chmod` always assumes that the special access class is 0. Therefore, if you set an suid-root file to mode 777, `chmod` assumes that you meant mode 0777, which would take away the special permissions. This is consistent if you remember that octal modes represent *absolute permissions* and *the special group class is assumed to be 0 if a 3-digit mode is provided*.

The following is a table to help calculate permission modes:

chmod absolute values											
special			user			group			other		
s	g	t	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1

For example, full access is 0777, which represents:

special			user			group			other		
s	g	t	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1
0			7			7			7		

There are no special bits set, so the `special` octal digit is 0, while all three bits in each other class are set. Each set of bits totals 7 (4 + 2 + 1), so the mode is 0777.

Another example is mode 2755, which represents `setgid` (2), plus "full access" for `user` (4 + 2 + 1) and `write` and `execute` (4 + 1) access for both the `group` and `other` class.

special			user			group			other		
s	g	t	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1
2			7			5			5		

Absolute modes are applied like this:

```
$ chmod 0755 somefile.sh
```

Absolute mode is great for setting things to be exactly what you want, or imposing a radically different order onto a file or directory, but it's not very good for adding the sticky bit to a file. For that kind of work (or if the octal modes just confuse you) the symbolic mode is well suited.

Symbolic -- more user-friendly

The symbolic mode works pretty much the way you would expect. To add the `execute` bit to the `user` class, you would execute a command like this:

```
$ chmod u+x somefile.sh
```

Or to add `execute` to all three classes:

```
$ chmod ugo+x somefile.sh
```

To make a file `setuid`:

```
$ chmod u+s somefile.sh
```

To remove all permissions:

```
$ chmod ugo-rwxsgt somefile.sh
```

Some people are so put off by the absolute mode that they never learn it -- you'll find with experience that both methods of setting permissions can be expeditious depending on the situation. More information is available in the `chmod` manpage.

Regardless of how you set permissions, it is critical that you use the "principle of least privilege" and only grant the privileges that are necessary for proper operation.