October 5, 2021

Commands

chmod

In this meeting, we briefly went over file privileges (see the end of this file) before learning how to change them using `chmod`.

You can quickly make a file executable by running `$ chmod +x filename`. This will allow you to run it by doing `$ ./filename`. Useful for if you download an executable but run into the `permission denied` error when you try to run it; Just use `chmod` to give yourself permission!

If you want more control over the privileges you assign, you can use a series of 3 numbers in place of the `+x` (1 number for each permission group). Here's how it works:

Each privilege you want to add to a specific user (Read, write, execute) is represented by a bit (1 or 0, on or off). For example, if I want `Read` and `Execute` privileges, I would put 1 for `Read`, 0 for `Write`, and 1 for `Execute`. This turns on `Read` and `Execute` privileges while turning `Write` privileges off.

So, I have 1, 0, and 1. Put those together and get 101. This is the binary representation of 5, so the number you would use to set this specific combination of privileges would be 5.

Now an example: If I want to give the owner `Read`, `Write`, and `Execute` privileges, the group `Read` and `Execute` privileges, and everyone else no privileges, I do the following process:

```
Owner: Read (1), Write (1), Execute (1) -> 111 -> 7
Group: Read (1), Write (0), Execute (1) -> 101 -> 5
Other: Read (0), Write (0), Execute (0) -> 000 -> 0
```

Now I can change the privileges of a file using `$ chmod 750 filename`

strings

Binary files, programs, text files… They all have one thing in common: There might be some random English words scattered around inside them. Instead of opening the file and searching through them, use the `strings` command which does that for you.

`$ strings filename` will print out all the human-readable words in the given file. Very useful, and we recommend running `strings` on every binary file you get, as soon as you get it. You never know, it might print out the flag right then and there.

exec

Allows you to execute terminal commands in a program. For example, you could type `ls` on the terminal and get a list of files, or you could write a program that does `exec(ls)`, which does exactly the same thing.

Challenges

Static ain't always noise

https://play.picoctf.org/practice/challenge/163

Given a binary file called `static` and a script called `itdis.sh`. We used `chmod` to give ourselves execute privileges for the script, then ran it on the binary file: `$ ./itdis.sh static`. The output was a bunch of random strings, so we used `grep` to narrow down the output to only include strings with `pico` in them (the flag is of the format picoCTF{flag}): `$ ./itdis.sh static | grep pico`.

This printed out the flag for us.

Tab, Tab, Attack

https://play.picoctf.org/practice/challenge/176

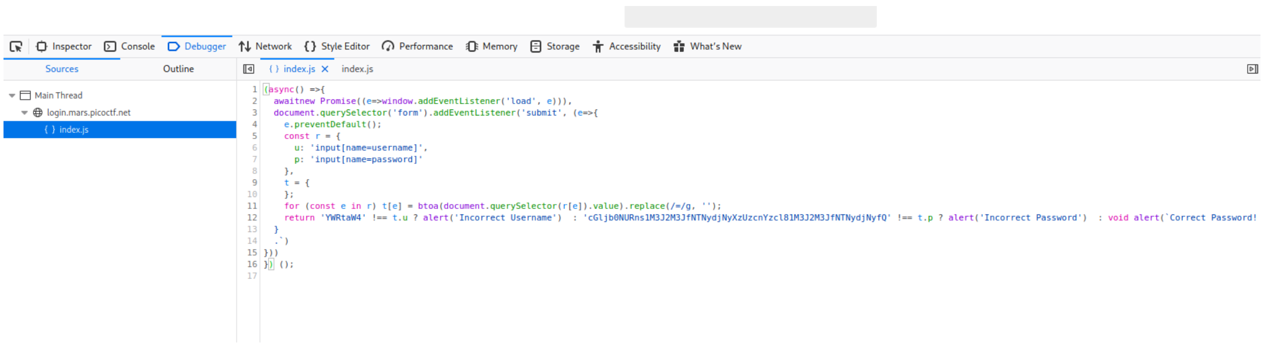
This is a super simple one, just practicing using the tab key to autocomplete directory and file names so we don't have to type them manually. Download the provided `.zip` folder, use `unzip filename` to extract the contents, then navigate to the program placed inside 5 or so nested directories. Use the tab key to be speedy.

Run the program, or run `$ strings program_name`, to get the flag.

Login

https://play.picoctf.org/practice/challenge/200?originalEvent=67&page=1

We're given a webpage, so the first step is to open developer tools and start poking around. We took a look at the html, looking for any comments or other easy pickings, but there were none. Moving on to the JavaScript, something suspicious was found:



```
(async() =>{
  awaitnew Promise((e=>window.addEventListener('load', e))),
  document.querySelector('form').addEventListener('submit', (e=>{
    e.preventDefault();
    const r = {
      u: 'input[name=username]',
      p: 'input[name=password]'
    },
    t = {
    };
    for (const e in r) t[e] = btoa(document.querySelector(r[e]).value).replace(/=/g, '');
    return 'YWRtaW4' !== t.u ? alert('Incorrect Username')  : 'cGljb0NURns1M3J2M3JfNTNydjNyXzUzcnYzcl81M3J2M3JfNTNydjNyfQ' !== t.p ? alert('Incorrect Password')  : void alert(`Correct Password!
  }
  .`)
}))
}) ();
```

It checks if the password is equal to a given string, and allows access if it is. So, we tried putting those values (`YWRtaW4` and `cGljb0NURns1M3J2M3JfNTNydjNyXzUzcnYzcl81M3J2M3JfNTNydjNyfQ`) in as credentials, but that didn't work.

Then, as a complete random guess, we put the password through a base64 decoder. We had seens base64-encoded strings before, and this looked similar. And wouldn't you know, that was exactly right!

Sometimes that's just how she goes.

caas

https://play.picoctf.org/practice/challenge/202?originalEvent=67&page=1&search=

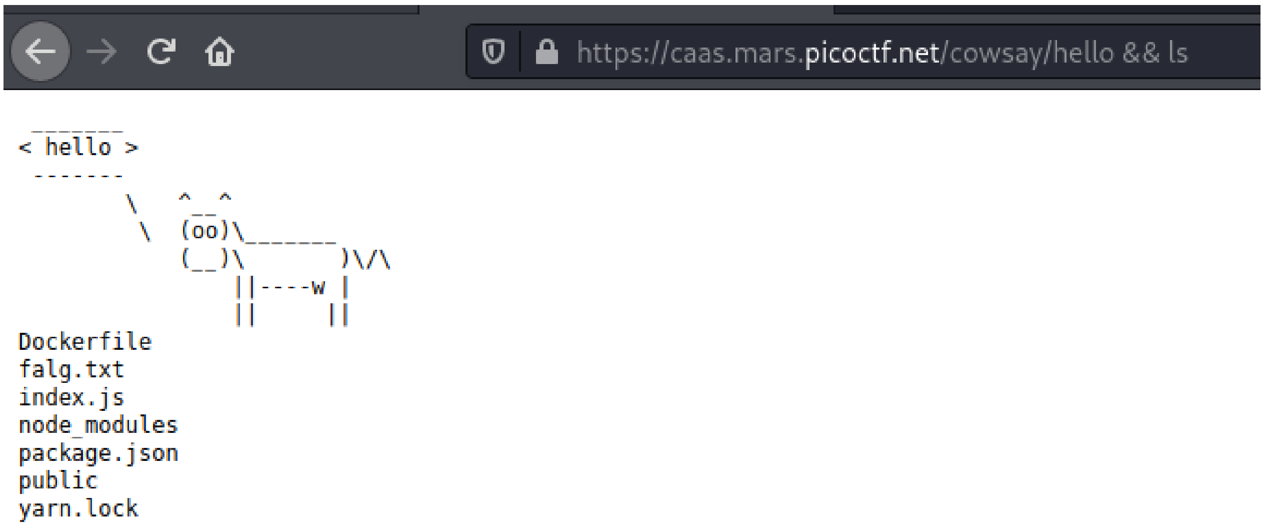
For this one, we're given a webpage and an `index.js` file, which is the code that runs the webpage. We started off by looking at the JavaScript file. It is very small, but contains an important function call: `exec`. More importantly, the input to the `exec` call is taken directly from the URL. We control the URL, therefore we control the `exec` command.

We're trying to find a flag, which is probably in a file somewhere. So, we want to execute `ls` to give us a list of all the files on this website's server. The code tells us that whatever we supply, the program will say, but we don't want it to print our command, we want it to be executed.

To acheive this, we can use special characters like `&&` and `||` to separate our input into two commands. On the terminal, if you type `ls && echo hello`, it will print out all the files in your current directory, then immediately after it will print 'hello'. The use of `&&` allows us to execute multiple commands in one line.

We can use this in the URL to split our input into two commands. The first word will be used as input to the `/usr/games/cowsay` program, and our second will be whatever command we want to execute.

`https://caas.mars.picoctf.net/cowsay/hello && ls` will have the cow say 'hello', then the program will print out all the files on the server.



```
https://caas.mars.picoctf.net/cowsay/hello && ls
```

```
 _______
< hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
Dockerfile
falg.txt
index.js
node_modules
package.json
public
yarn.lock
```

Look at that, a file called `falg.txt`, that's probably what we're looking for. We can then use the `cat` command to print out the contents of a file, giving us the flag.

`https://caas.mars.picoctf.net/cowsay/hello && cat falg.txt`

Extra Stuff

File Privileges

All files have privileges assigned to them, `Read`, `Write`, and `Execute`. Someone viewing the file may have `Read` permission (they can view the file), but they cannot change it (no `Write` privilege).

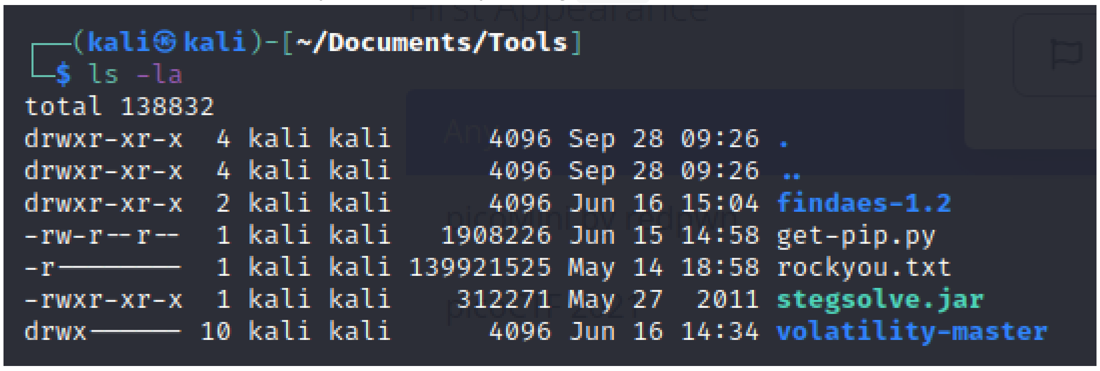
There are also 3 `permission groups` that each file has: `Owner`, `Group`, and `Other`. Every file has an owner and a group assigned to it, and each permission group has their own privileges (`Read`, `Write`, `Execute`).

For example, imagine I own a file called `file.txt`. I want to be able to read and write to it, but I don't want anyone else to modify it. Everyone else can read it, they just can't change it. I would set `Read` and `Write` privileges for the `Owner`, but only set `Read` privileges for `Other`.

But what about `Group`? Imagine I had a few friends that I wanted to give write access to. I couldn't add `Write` access to `Other`, because that would allow *anyone* to change the file. Instead, I make a new group (a tutorial for another time), add my friends to that group, then give write access to that group.

After all that, `file.txt` can be read and modified by my friends and I, but everyone else can just read it.

You can see the permissions all your files have by running `ls -la` in a terminal window.



```
┌──(kali㉿kali)-[~/Documents/Tools]
└─$ ls -la
total 138832
drwxr-xr-x   4 kali kali      4096 Sep 28 09:26 .
drwxr-xr-x   4 kali kali      4096 Sep 28 09:26 ..
drwxr-xr-x   2 kali kali      4096 Jun 16 15:04 findaes-1.2
-rw-r--r--   1 kali kali   1908226 Jun 15 14:58 get-pip.py
-r--------   1 kali kali 139921525 May 14 18:58 rockyou.txt
-rwxr-xr-x   1 kali kali    312271 May 27  2011 stegsolve.jar
drwx------  10 kali kali      4096 Jun 16 14:34 volatility-master
```

Notice the `drwxr-xr-x` on the left. This is a string of characters representing the privileges we just went over. `r` for `Read`, `w` for `Write`, and `x` for `Execute`. The character `-` means that that privilege is not set. If you then group the characters in groups of three, starting from the right, you get your 3 different groups:

```
    (owner) (group) (other)
d     rwx     r-x     r-x
```

`d` means you're looking at a directory. If the `d` is not there, you're looking at a file.

We can now see that the Owner has read, write, and execute privileges, the group has read and execute privileges, and so does everyone else.

If you look at the row for the `volatility-master` folder, you can see that the owner has read, write, and execute privileges, but the group and others have no privileges. They can't even open this folder!