G53SEC LAB 4: PASSWORDS

LAB DESCRIPTION

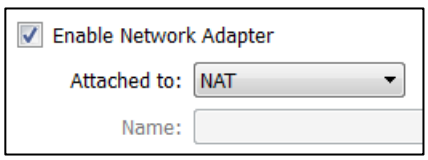
In this lab you'll be familiarizing yourself with the password system in Linux. Adding users, supplying a password, and allowing them access to the sudo group, to avoid them logging in as root. You'll also control the password policy, which affects what users can choose as passwords. Last but not least, you'll learn how to use one of the world's most advanced password recovery applications - Hashcat. There are 6,400 passwords available to crack in this session, how many will you manage? Perhaps more importantly, will your own passwords yield?

INTRODUCTION

Historically passwords have been the weak link in countless system breaches worldwide. Passwords are sometimes guessed (e.g. defaults left unchanged), derived from known information about the target, or obtained through phishing. In many cases passwords are simply stolen following the breach of a larger system. Usually (although not always) passwords are encrypted with a one-way hashing function. This serves to obscure the password from prying eyes, while still allowing authentication of the user. Although these hashes cannot be reversed, on weak passwords it is trivial to simply hash numerous words until you determine the password. Even strong passwords will often yield to an efficient attack. This makes strong password policies more vital than ever.

LOGIN INFORMATION

A description of the Kali operating system and the setup of the virtual machines is given the introduction document. A small part of this lab will require you install software, which will require your machine be online. By default the Kali VM may not be online when you start it up, so before you do, go to Settings, Network, and from the "Attached To" list select "NAT"



Below are the login details for reference, as always you should avoid logging in as root and use sudo to elevate your privilege if needed.

If we see you logged in as root while we're walking around the labs, we will call you on it.

Normal User
Username: sec
Password: security

Root
Username: root
Password: toor

LINUX PASSWORDS

Much like other operating systems, on the surface Linux's password system is straightforward to use. Any user can manage their password from the terminal, and the root user can change the password of any other user. The primary utilities for adding and modding users are, perhaps unsurprisingly, `useradd` and `usermod`. Let's begin by creating a new user and providing them a password.

```
sec@kali:~$ sudo adduser uri
```

You'll be prompted for a password for uri, and some other details – enter whatever you choose. If you run `groups uri` you will see that by default the new user has been put in their own group. This serves as the default group that will be used when they create files and perform other tasks. A user can be in multiple groups, more on that soon.

Let's test out uri's account: Click the power icon in the top right, select your username and choose to switch user. Login as uri, then head to the terminal and look around. You'll see that uri has a number of directories within their home directory. There are more than sec has, because I deleted some of the more useless ones before I distributed this VM! Anyway, let's try and do a couple of things as uri:

```
uri@kali:~$ cd Documents
uri@kali:~/Documents$ echo "Hello World" > file
uri@kali:~/Documents$ cat file
Hello World
```

Let's get a bit more adventurous:

```
uri@kali:~$ cp /etc/shadow ~/Documents
cp: cannot open '/etc/shadow' for reading: Permission denied
uri@kali:~$ sudo cp /etc/shadow ~/Documents
...
uri is not in the sudoers file. This incident will be reported.
```

Worth a shot! Obviously uri is neither a root user, nor a user in the sudo group. Let's change this – they seem trustworthy. Log back in as sec. Modify uri's groups like so:

```
sec@kali:~$ sudo usermod -a –G sudo uri
sec@kali:~$ groups uri
uri: uri sudo
```

Have a look at the manual page for usermod. The `-G` flag instructs it to add an existing user to a group, the `-a` option instructs this is an append operation, i.e. we want the user to stay within their existing groups as well. If you don't see `uri` and `sudo` as groups, use `usermod` to fix this.

Finally log in as uri and see if you can pinch the Linux shadow file this time. You will probably need to have logged out and back in as uri (rather than switch user) to see the changes you've made.

As a final note, remember you can change the password for any user as a sudoer, using `sudo passwd username`.

PASSWORD POLICIES

Given that uri is now a sudoer, a breach in their account is as serious as a breach in the root account. A password policy is a set of rules that govern what types of passwords you can use, and how often they need to be changed. For example, rules such as:

- Passwords must be at least 8 characters
- Passwords must contain at least one upper-case character, symbol and number
- Passwords must be changed after 90 days
- Warnings begin that a password should be changed 7 days prior to expiration
- After 7 days with an expired password, an account is locked

In Linux, a root user can see the password expiration policy of any user using either the `passwd` or `chage` commands. `chage` is a more usable command in general, so we shall focus on this. Log back in as sec, then have a look at uri's password policy (yours will obviously be a little different).

```
sec@kali:~$ sudo chage -l uri
```

This will be self-explanatory. Have a look at the manual page for chage, then set an expiry date for uri's password for a couple of days' time. If you complete this correctly, when you log in as uri, a (very) brief warning should appear. You can also check `chage -l` to ensure the correct date has been entered.

PASSWORD COMPLEXITY

Expiring passwords is one way to ensure that compromised passwords are removed from the system, but it doesn't deal with the root cause of the problem, weak passwords. Users will still, despite efforts to educate them, use ridiculously weak passwords. If these users are in the sudo group, we have a problem!

Kali, and now other similar Linux distributions, come with basic password checks that stop users choosing very weak passwords. Try to change the password for sec now, try out a short

password, and a longer palindrome. For example “bad” and “racecar”. Passwd will give up trying to change your password after 3 or so attempts, at which point “security” will remain the current password.

These efforts to prevent very weak passwords are a good start, but don’t go far enough. “security” is not a strong password; it’s not long, it is purely lowercase letters, and it’s a dictionary word. Hashcat would crack this password in seconds. Using more powerful tools, we can coax users into choosing better passwords.

Linux uses a Password Authentication Module (PAM) to verify passwords. Basically, most programs in Linux ask PAM to verify passwords when they need authentication. The standard module that’s installed on this distribution is `pam_unix`, which is what performs the basic checks above. Have a look at the manual page for `pam_unix`, in particular look at the options, which list some of the different checks we can enable. By default, the “obscure” check is enabled, as is the SHA512 hashing function.

All the settings for PAM are stored in the `/etc/pam.d/` directory, try listing these files now. Any programs that use PAM for authentication will have a file in here. Most, however, will simply reference `/etc/pam.d/common-password`. Have a look at this file now, find the line that details the settings for `pam_unix`.

```
password    [success=1 default=ignore]    pam_unix.so obscure sha512
```

This and the lines below detail the rules PAM follows to authenticate users. Notice the obscure and sha512 keywords.

Now let’s install a more powerful module, `libpam-cracklib`. Packages are installed in Debian variants using dpkg or the apt-get front end. Let’s install cracklib now:

```
sec@kali:~$ sudo apt-get update
sec@kali:~$ sudo apt-get install libpam-cracklib
```

This will make various changes to the OS, including installing the binaries and settings, and updating the manual database with the relevant help files. Re-open common-password and take a look at the changes:

```
password    requisite         pam_cracklib.so retry=3 minlen=8 difok=3
password    [success=1 default=ignore]    pam_unix.so obscure use_authtok
                                              try_first_pass sha512
```

You can see that cracklib has now inserted a few more parameters, and also instructed pam_unix to use its output. You can look at the manual for pam_cracklib to see details on the different attributes, or see here. In particular minlen is the minimum length of the password, although it will allow the password to be shorter if you “earn credits” by using more varied characters. Set this minlen to 12, then save and close the document.

Try changing your password again, you'll see it's stricter this time, dictionary words can't be used, and it enforces this length constraint. Once you've tried this out, change your password back to "security" for now. You can't do this the normal way, because security is a dictionary word, but if you use sudo, you can force change the password. Cracklib will still complain though!

For interest, there are many other authentication modules you can use, for example libpam-google-authenticator, which will add Google's two-step authentication process. It's possible to install this for ssh access, but have regular authentication once you're already logged on.

CRACKING PASSWORDS

As we will cover in lectures, passwords are usually stored in a hashed form, so that if they're stolen (or simply viewed by a nefarious administrator with root access), they can't be read. This is a pretty simple process, the MD5 algorithm will turn any string into a fixed string, 128 bits in length. SHA512 will do the same thing, outputting 512 bits. For example:

```
sec@kali:~/lab4$ echo –n "password" | openssl md5
(stdin)= 5f4dcc3b5aa765d61d8327deb882cf99
sec@kali:~/lab4$ echo –n "password" | openssl sha512
(stdin)= b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb98
0b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86
```

These hashes are one-way functions. Unlike standard encryption, you can't reverse the process. Password cracking involves simply trying a large number of possible passwords, hashing them, and seeing if they match the stored password hash. If they do, we have the correct plaintext password.

When MD5 was designed, it was unforeseeable that computers would be fast enough to brute-force crack passwords stored in it. As you will soon see, this resilience has been greatly overestimated.

The passwords for this unix installation are stored in the sha512 hash format, within the /etc/shadow file. We won't be cracking these passwords today, although by all means have a stab if you want to try. Instead we'll be using a small database of hashed passwords that are supplied with hashcat. These passwords are hugely varied, and range from utterly trivial through to essentially un-crackable. You can look at the hashes within the ~/lab4/hashes/hashdb file if you're interested, they are the result of the MD5 algorithm.

HASHCAT

Hashcat is an open-source password cracking tool that can implement a huge variety of hashing functions, and a huge variety of password guessing techniques. Since we're running in a Virtual Machine, we can't utilize Hashcat's GPU capabilities, but this CPU edition is still very effective. If you'd like to try Hashcat on your GPU at home, by all means, it is brutally effective.

What I'll do here is provide a brief overview and example of the common ways to use Hashcat, then you should just get stuck in.

LAB 4 FILES

Let's quickly look at the resources available in the `~/lab4` directory.

- `hashes/` : Folder containing the hashed passwords for cracking, and a backup version
- `dicts/` : Folder containing two dictionaries of common passwords
- `rules/` : Folder containing common password manipulation rules
- `pack/` : Password analysis tool, we will cover this later
- `hashcat.pot` : This might not exist yet, it is the file in which hashcat will store recovered passwords

There are also some scripts I have provided for utility reasons. If you'd like to know more about how these work, open them up and look, or ask me in the labs. The scripts are:

- `status` : Compares the sizes of hashcat.pot and hashes/hashes to determine what percentage of passwords you've cracked
- `reset` : Deletes all currently cracked passwords and restores the list from backup. Essentially resets your cracking efforts.
- `submitpasswd` : Prompts for a password, then hashes it and appends it to the hashes file
- `checkpasswd` : Prompts for a password, then will print a message telling you if it is inside the pot file.

OPTIONAL

The last two scripts let you test the resilience of your own passwords, without actually having to show them to people in the labs.

You can insert a password into the list using the `./submitpassword` script. It will obscure the password, so you can use this script in front of people. Once some cracking has occurred, if your password was cracked it will have found its way into the hashcat.pot file. Obviously you don't want to directly search for it, so use ./checkpassword to see if it's in there.

Individuals and groups can submit any number of their passwords this way.

The general approach to cracking is as follows:

1) Run `hashcat` with various parameters, including the `--remove` flag. This removes them from the hash file after they have been cracked, preventing you having to repeat them.
2) Run `./status` to see how you are getting on

3) Repeat!
4) If you want to try again from scratch, use ./reset to restart. This will also delete any custom passwords you have entered.

ALL THE PASSWORDS

Hashcat has various modes (view using `hashcat --help`), traverse to the lab4 directory and let's get started.

Mask Attack – Attack Mode 3
The successor to brute-force, this technique works by trying every combination of passwords using a given character mask. The command is:

```
hashcat –a 3 --remove hashes/hashdb [mask]
```

The mask is a string containing the types of characters / digits you'd like to try out. These are:

?l lowercase
?u uppercase
?d digits
?s symbols
?a all of the above

If you'd like to try every combination of 6 character lowercase passwords (e.g. aaaaaa, aaaaab, …, zzzzzz) then you would use the following command:

```
hashcat –a 3 --remove hashes/hashdb ?l?l?l?l?l?l
```

If you run this, it'll take just over 10 seconds, and successfully crack a few passwords! Successful cracks are output to the screen as hash:plaintext. They're also stored in the hashcat.pot file.

Try a few mask attacks. Think about what kind of format people might use, perhaps a couple of digits at the end? The longer the mask, and the more complex the character set, the longer this will take. Mask / brute force attacks are meant to find the super-weak passwords quickly. We have better tools for better passwords.

Straight Attack – Attack Mode 0
The straight attack literally reads from a dictionary file, trying out all of the passwords therein. This is much more effective than you think when using a good dictionary, common passwords (qwerty etc.) are easily beaten this way.

We have two dictionaries available, both in the `/dicts` folder. Let's start with a small word list:

```
hashcat –a 0 --remove hashes/hashdb dicts/small.txt
```

Fast, but not hugely effective, this dictionary just isn't long enough. It'll be more effective later when combined with rules. The other dictionary we have is a lot more powerful. In 2009 the RockYou gaming service was hacked, exposing 32 million un-hashed passwords. Since then, this list has become the standard dictionary for straight, rule and combination attacks. Re-run the attack with the rockyou list, and watch the passwords fall.

Combinator Attack – Attack Mode 1
You can magnify the power of a password dictionary using a combinator attack. This combines a dictionary with itself into all the combinations of appended passwords. For example, the following dictionary:

```
pass
1234
lol
```

Will produce `{ passpass, pass1234, passlol, 1234pass, 12341234, 1234lol, lolpass, lol1234, lollol }`

Again, you'd be surprised how many people chose passwords by just putting 1234 on the end of things. Try this attack with the small dictionary:

```
hashcat –a 1 --remove hashes/hashdb dicts/small.txt
```

Now try it with Rockyou, it's incredibly effective, though depending on time constraints you may have to quit this one early!

Rule Attack – Attack Mode 0, Rules
Like with the combinator attack you can magnify the cracking power of a dictionary by messing with the passwords. For example, if qwerty is a password in your dictionary, it seems likely that someone might simply put a space in front, and " qwerty" is worth an attempt. Similarly, trying a few random digits on the ends of common passwords is extremely effective, or replacing e with 3, and i with 1. Try this:

```
hashcat -a 0 --remove -r rules/best64.rule hashes/hashdb dicts/small.txt
```

This takes all of the rules in the best64.rule file (which is the best attempt at a concise rule list from the developers and community) and applies them to each word in the dictionary. Take a look at the rule list, some are intuitive, reverse words, append 0, append 1, etc. Some are more complex, in any case, it works. Try with the rockyou database.

GOING IT ALONE

You have almost all the tools you need now to break the majority of these passwords. I've managed 75% at best, the last 25% might take a while! Still, that's over 5000 passwords that, were I malicious, I could use to gain access somewhere or perform phishing. Now you're on your own, try and crack as many passwords between you as you can. Some tips:

- Use mask attacks, but only to a point. They get slow quickly as you increase the mask length, find the low hanging passwords and move on. Try to think about the kinds of structure people might use, which letters do they capitalize? Where do they put digits usually?
- Rule based attacks offer (in my opinion) the fastest rate of cracking of all of these methods. That's not to say you should use them alone, but explore the rules directory, what combinations of rules and dictionary is effective? While the small.txt dictionary is weak compared with rockyou, it also contains simple passwords for rule manipulation. Many of rockyou's passwords are very obscure.

Password Analysis

If your cracking rate has started to slow down, maybe you've hit 1-2000 passwords, and getting more is proving hard, the Password Analysis and Cracking Toolkit may help. This tool analyses the passwords you've already found in an effort to determine the kind of masks that may yield more. For example, are 90% of the users producing passwords than end in two digits?

To use this tool, which is stored in the `/pack` directory you'll first need to separate the hashes from their plain text passwords in hashcat.pot. PACK requires a list of passwords only. This command will do that:

```
cat hashcat.pot | awk -F':' '{print $2}' > plainpasswords
```

This puts all of the plain passwords from hashcat.pot into plainpasswords. Then you can call

```
python ./pack/statsgen.py plainpasswords --hiderare
```

If you've cracked enough passwords for reliable statistics, you might notice a lot of passwords seem to be loweralphanum, i.e. lowercase letters and numbers. Perhaps you can develop masks with these statistics in mind.

For documentation of this tool, and others in the toolkit, see here. It will even generate rule and mask files for future cracking. Bearing in mind this would be most effective when run on a GPU overnight.

Conclusion

In this lab session you've familiarized yourself with Linux password management and policies. You've then gained some experience of recovering passwords, despite them being protected by a hashing function. Hopefully your own passwords weren't cracked by the software, but regardless, the way the password cracking software works should give you a good insight into which passwords are secure, and which are not.