KIT304 Server Administration and Security Assurance

Tutorial 19

Goal

The first half of this tutorial will be an exploration of password security. You will spend some time initially exploring password entropy, before moving on to attempting to break passwords in bulk. Following this, you will explore some of the tools that are available for conducting digital forensics.

Part 1 – Passwords

People use passwords every day. You use them to access university computers and services, your own computers, websites, software, and many other things. Password use is common because passwords are easy to understand, store, retrieve and use. Even those with very little computing experience understand the concept of a password.

People trust their passwords. They assume that no one knows them, that no one would guess them, and that through them they achieve their security goals. Unfortunately, for most people, this is not true. In many cases, passwords do not provide users with the security that they expect. Consequently, the use of other forms of authentication are becoming increasingly common:

- the inclusion of biometric-based authentication mechanisms in laptops and phones has grown significantly in the last 10 years
- online services increasingly send SMS or in-app codes that users must enter to gain access
- many services support or even require the use of one-time codes generated by an authenticator application running (typically) on a mobile device to confirm the user's identity

In this tutorial you will explore how effective (and weak) passwords are, and the techniques that can be used to increase password strength and therefore increase the level of difficulty and amount of work required for a bad actor to be able to crack or discover them.

Password Strength

Passwords are usually short strings of letters and numbers. Many people, before they understand the nature of passwords and the effectiveness of tools that can be used to obtain (or crack) them, use passwords that are easy to remember, and/or that relate to them in some way. They may be dates (birthdates, anniversary dates, and so on), a pet's name, a family member's name, a character

or place name from a favourite book or movie, a sports team name, a phone number, or something similar that has a special meaning for, or is something that the person cares about.

All of these tend to be either real words (words that are most likely stored in a dictionary, or at least in a commonly used password dictionary that an attacker may use), or things that an attacker using social engineering techniques can discover about the user relatively easily.

The important point is that these types of passwords are predictable, or guessable. Instead of being something that is secret and hard to guess, they are often the direct opposite of that.

Other than passwords that relate to us in some way, some words are used very commonly as passwords. CNN Business reported on April 23, 2019 that the UK's National Cyber Security Centre (NCSC) had analysed passwords belonging to accounts worldwide that had been breached. The most common passwords were:

1. 123456
2. 123456789
3. qwerty
4. password
5. 111111
6. 12345678
7. abc123
8. 1234567
9. password1
10. 12345

Others in the top 20 included **iloveyou**, **monkey**, and **dragon**.

Humans can be unoriginal at times, and password choices are one of the situations where this is noticeable and indeed measurable, as we will see later in this tutorial.

The result of these biases (to themselves, or to passwords that people choose generally) is that the passwords many people generate aren't *random*. If a person chooses a password that is seven characters in length, the goal is to select one that is secret and secure. However, any degree of predictability results in a password that is the opposite of random. A truly random password would include any combination of letters, and therefore be much harder to predict. If every one of those 7 characters could be any letter from alphabet, the number of combinations would be:

$$\mathbf{26 \times 26 \times 26 \times 26 \times 26 \times 26 \times 26 = 26^7 = 8{,}031{,}810{,}176\ (8\ Billion)}$$

If you include a mix of upper- and lower-case letters, the number of possible combinations is even higher:

$52 \times 52 \times 52 \times 52 \times 52 \times 52 \times 52 = 52^7 = 1{,}028{,}071{,}702{,}528$ (1 trillion)

Furthermore, if the password can include any character commonly available on a standard computer keyboard, that includes:

- 52 alphabetic characters
- 10 numeric digits
- 10 symbols on digits
- 22 other easily entered symbols
- 1 space

This is a total of 95 easily typed characters that could be used as part of a password.

A 6-character password, where each character can be one of 95 possibilities, allows for 750 billion combinations - that is 75% of the number of passwords that are available with a 7-character password using just upper- and lower-case letters.

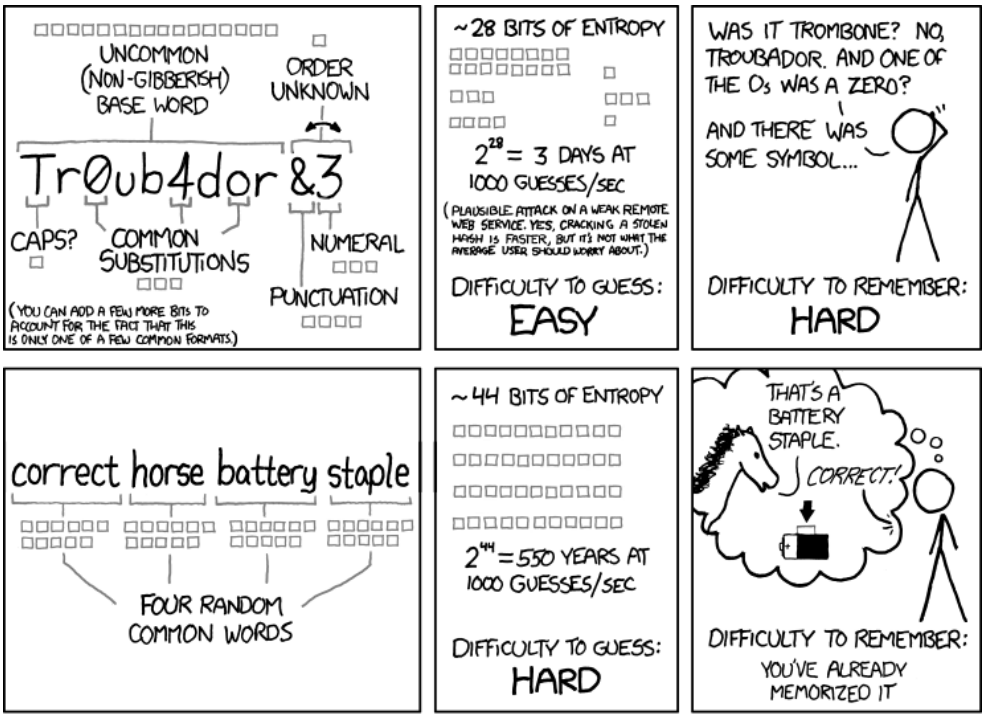
A 7-character password where each character can be one of 95 possibilities allows for over 70 trillion combinations (this is around 70 times the number of possibilities compared to using alphabetic characters alone), and an 8-character equivalent provides over 6.5 quadrillion combinations.

This is a lot of passwords. The randomness of a password, or a key in a cryptographic system, is discussed in terms of *entropy*. Entropy is the measure of randomness. If you choose a predictable password, or use a dictionary word, it is said to have *low entropy*.

As discussed earlier, people don't create passwords from random characters. Ideally, passwords like **c3e%{_?L**, which have *high entropy*, aren't used because people find these hard to remember and difficult to type. The result is that people write passwords on post-it notes, and stick them on monitors or under keyboards, which reduces security.

One solution to the complexity vs familiarity issue is the subject of the **xkcd** comic on the next page. Instead of using shorter hard-to-remember passwords, one way forward is to increase the length of the password as a whole, even if that means it is composed of components that *are* easy to remember. The longer the password, and the more components that are used to create it, the harder it is to guess using brute force methods.

Another practical solution is to use a password manager like **LastPass**, **One Password**, **KeePass** or **Dashlane**. These tools don't work in all situations, and don't suit everyone – they aren't common in the mainstream, and are mainly only used by technically knowledgeable people, but they do support the creation of very long random passwords that have high entropy, aren't associated with the user, and are at the high end of difficulty when it comes to password cracking.



https://xkcd.com/936/

Now that we have had our introduction about passwords, and what they are made of, we will talk briefly on storage before then for the remainder of the tutorial attempting to break some passwords.

Password Storage

Back in Tutorial 15, during the penetration testing exercises, you broke into an account over the local network using the **hydra** application, which tried hundreds of passwords to brute-force a login.

Once you logged in, you looked at the file **`/etc/passwd`** – the standard place where Unix systems store a list of authorised users of the system. No passwords are stored in this file – instead, they are stored in the file **`/etc/shadow`** – one password for each user in the **passwd** file. These passwords have been *hashed* – they are not stored in clear text for anyone to read. When a user attempts to log in, the system hashes the password that they enter, and then compares the generated hash with the hash stored for that account in the **shadow** file. This means that if anyone is able to steal the **shadow** file, they don't have the actual passwords, just the hashes that represent them.

This is the standard for password storage security: don't store the passwords in plain text, but rather store them in a hashed form. This applies not just to operating systems, but also to websites and other online services that support user logins. Every year millions of user accounts are stolen from insecure online services. These are stolen in a hashed form most of the time (unless the service uses very poor security practices). Today you will be seeing how many passwords you can break in a few simple queries on one such stolen database.

Activity 1 – Breaking Hashes with Password Dictionaries

Your first activity will include examining a file that contains the hashed passwords to over 500,000 accounts. The account names and passwords were stolen in a hack. The hashes from this hack are stored in the **Tutorial 19** folder, which is inside the **KIT304** folder on the **MacOS** Desktop in a file called **`bfield.hash`** (the account names and other information has been removed). Open the file and look at the hashes. You should be able to open it in **TextEdit** or **BBEdit,** or even **vi** in the terminal.

Back in Tutorial 15 you used a simple list of passwords to break into one of the target computers on the virtual network. You will use this file again as the first list of passwords to test against the target hashes. It was called **`passwordList.txt`**, and is also in the **Tutorial 19** folder. Feel free to open it again and view the contents.

There are many different applications for discovering which password produces a hash. On **Kali** Linux there are a handful of different tools for doing this. The one you will use today is called **`hashcat`** and is one of the most popular tools for breaking a hash. A Mac version is also located in the **Tutorial 19** folder.

Note that you'll be completing this exercise on the Macintosh, not on one of the virtual machines.

1. Copy the files **`bfield.hash`** and **`passwordList.txt`** into the **`HashCat`** folder.

2. Launch the macOS terminal from the Dock at the bottom of the screen. In the terminal window, change to the **`HashCat`** folder with this command (use **tab-completion** at each folder name to reduce the amount of typing) – note that as you do this you may a security warning popup stating that the Terminal would like to access files in the Desktop folder. If you do, you need to click OK to proceed:

   ```
   cd Desktop/KIT304/Tutorial\ 19/HashCat
   ```

   Once in the folder, use the **`ls -al`** command to ensure that the two files you copied earlier are there.

3. To learn how to use **`hashcat`**, enter the following command:

   ```
   ./hashcat-cli64.app -h | more
   ```

   As you can see, it has a large number of options and modes that can be employed in an attempt to crack hashed passwords.

4. To use the small list of passwords (**`passwordList.txt`**) against the **`bfield.hash`** file enter the following command:

```
./hashcat-cli64.app -m 0 -a 0 bfield.hash passwordList.txt
```

What **hashcat** is doing here is generating MD5 hashes of every password in **passwordList.txt,** and then comparing that hash to entries in the bfield.hash file. Thus, it is able to determine the original password for many of those hashes.

How many passwords did it discover from within the 548k? _____

5. Now, you'll repeat this process with a much larger list – a dictionary of English words. Back up one level in the Tutorial 19 folder is the file **english.txt** – open this and view its contents, copy this file into the **hashcat** folder, and then enter the following command:

```
./hashcat-cli64.app -m 0 -a 0 bfield.hash english.txt
```

How many did it find this time? _____

This shows that while using hashes for passwords is better than storing them in plain text, if the hashes are not secured properly, many of the passwords can be recovered using tools like **hashcat**.

In December 2009, a hacker took advantage of an SQL injection vulnerability and stole 32 million passwords from **RockYou** – a popular application developer on Facebook and MySpace. Those passwords were stored in plain text in an unencrypted database, and included email addresses. As discussed earlier, many people use dictionary words as passwords, and users frequently reuse passwords across multiple sites. Thus, anyone in receipt of the data from the breach could trivially attempt to gain access to each user's email account using their RockYou password, and in many cases those passwords would have worked.

One of the simplest options for discovering which password each hash represents is to not just try words from a dictionary, but to also try passwords obtained through other successful breaches.

6. Open the **rockyou.txt** file (also in the **Tutorial 19** folder) and look at the content, then copy the file into the **hashcat** folder and enter the following command:

```
./hashcat-cli64.app -m 0 -a 0 bfield.hash rockyou.txt
```

This will take a little longer to run. How many did it discover this time? _____

7. **hashcat** can do more than simply compare using a dictionary. It can use a variety of rules to build on the words already stored in the password file. Enter the following command - what do you notice that is different from the command from step 6 above?

```
./hashcat-cli64.app -m 0 -a 1 bfield.hash rockyou.txt
```

The **hashcat** command you're currently running will take some considerable time to complete – too long to wait for our purposes today. So type a Control-C in the terminal window to cancel it, and move on to the next activity.

Activity 2 – Breaking Hashes with Rainbow Tables

As can be seen from the previous activity, it can take a considerable amount of time to break hashes. You didn't actually discover all the hashes in **bfield.hash** but it's possible to recover up to 98% of them – that is 7667 cracked hashes – with the right combination of options and some patience.

This level of detection takes a significant amount of effort, and using the approach in the previous activity, it involves creating a hash of each value we think might be a password and then comparing that value to the stored hashes. This means calculating a lot of hashes "on the fly". There is another popular approach that is the direct opposite to this.

A *rainbow table* is a pre-generated list that contains all possible hash values for a given hash method. This reduces the process of "cracking a hash" to a simple table lookup, and means that you can determine which password generated the hash very quickly. You don't need to calculate anything; you just look up that hash value. While this sounds easier, and fast, it does require a large amount of storage space for the exhaustive collection of hashes. There is a trade-off between the two approaches – speed vs storage space.

The KIT304 account's Windows 7 virtual machine includes a small rainbow table (around 3 gigabytes) that took approximately 4 hours to calculate on an 8 core CPU, with all cores running at 100%. The following table shows the amount of space required to store complete rainbow tables for a number of different password lengths and character styles using that same program. You might see from this why some people would rather purchase them (on a hard drive no less), or use them via a cloud service, rather than calculate them directly.

Set	Length	Key Space	Size
Lower alpha and Numeric	7	78,364,164,096	3.2 Gb
Mixed alpha and Numeric	7	3,521,614,606,208	15 Gb
Mixed, numeric and some special	7	70,576,641,626,495	52 Gb
Lower alpha and Numeric	8	2,821,109,907,456	13 Gb
Mixed alpha and Numeric	8	221,919,451,578,090	127 Gb
Mixed, numeric and some special	8	6,704,780,954,517,120	460 Gb
Lower alpha and Numeric	9	104,461,669,716,084	65 Gb
Mixed alpha and Numeric	9	13,759,005,997,841,642	690 Gb
Mixed, numeric and some special	9	6.30×10^{17}	?
Lower alpha and Numeric	10	3,760,620,109,779,060	220 Gb
Mixed alpha and Numeric	10	8.39×10^{17}	?
Mixed, numeric and some special	10	5.98×10^{19}	?

"Mixed, alpha and some special" includes the following: characters

```
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

8. In the VMWare Fusion Virtual Machine Library window, right click the **Win7** VM and choose **Settings**. Click on the **Processors and Memory** icon, and set the **Processors** menu item to **8 processor cores**. Then close the settings window, and launch the Windows 7 virtual machine. Once Windows 7 has booted, you'll most likely get a restart prompt from it as it tweaks itself for the higher CPU count that you configured. If you see this, click the **Restart Now** button and wait until it has rebooted.

9. On the Windows 7 Desktop in the **KIT304** folder, in the **Practical 17** folder is a file named **`bfield-cutdown.hash`** and another folder containing a utility for generating and using rainbow tables. Open this folder and run **`rcrack_gui`**.

10. You're going to use rcrack to search the previously calculated rainbow tables for hashes that match those in **`bfield-cutdown.hash`**. In the **rcrack** window, open the **File** menu, select **Load Hashes from File**, and then find and select the **`bfield_cutdown.hash`** file in the parent folder.

11. Next, go to the **Rainbow Table** menu and select **Search Rainbow Tables in Directory**. Browse to the **C:** and select the folder called **RainbowTables**. The program will now run, loading each rainbow table in turn, and searching for the hashes previously loaded. You may want to press Ctrl+Shift+Esc and view the **Performance** tab to see the resource usage of the search.

    This process will take around 9 minutes to run, and will start to find results after a few minutes – displaying each in the window alongside its hash as it does so. Once it has been running for a while you can move on to the activity below and check back periodically on its progress.

    When finished, how many passwords were cracked? _____

Since recalculating rainbow tables is a time-consuming and resource-intensive process, there are various online services that you can use that provide access to previously-generated tables. Crack Station (https://crackstation.net/) is such a website – it includes a 190 GB table of MD5 and SHA-1 hashes with 15 billion entries, and another table of 1.5 billion hashes that is 19 GB in size.

12. Browse to https://www.md5hashgenerator.com/ and enter a password that you would like to hash. You can enter any value you want, but you might want to try what you would now consider to be a weak password. Click the → **Generate** button to convert the password to an MD5 hash.

13. Next, open another browser window at https://crackstation.net/ and paste in the MD5 hash generated in the previous step. Return to the MD5 hash generator again, and enter another weak password. Paste this hash into the next line at Crack Station. Finally, go back to the MD5

hash generator again, and this time, enter a password that you would consider to be strong. Paste this final hash onto a new line at Crack Station.

14. On the Crack Station site, complete the reCaptcha on the right to prove you're not a robot, and then click the **Crack Hashes** button. Hopefully, Crack Station was able to reveal some of your weak passwords – was it also able to reveal what you considered to be a strong password? If so, you should re-evaluate your password generating techniques!

As can be seen by the two activities you've just completed, Rainbow Tables are a useful tool for cracking passwords. They are not necessarily better than dictionary approaches – they are different, with different trade-offs. The choice is between pre-calculating the hashes, and potentially requiring significant storage, vs. calculating them on the fly, potentially requiring significant time.

Salting Passwords

Both of the methods that you've employed to crack password hashes may have left you curious about how to defend against these sorts of threats once a system has been breached. What additional methods can be used to protect a password database?

From the user perspective, there isn't much that can be done other than to learn to use strong passwords, and not share them across sites.

From the perspective of a security professional, or a software developer, another best-practice recommendation is to use *salted passwords*.

Salt changes the nature (or more specifically, the flavour) of food. In password security, a salt is a fixed length sequence or random additional characters that are added to the start of the password before it is hashed. Salting effectively lengthens the password, but it also significantly diminishes the value of rainbow tables. Since the salt is randomised for each user, even if two users use the same password, there's almost no chance that the system will generate the same random salt for each of them, and so their hashed passwords will be different.

Since the number of salts is potentially infinite, it is impractical to generate rainbow tables for every salt variant. For example, if the salt was only two characters long, and those characters could be any of the 95 described earlier, then you'd need to generate 95 x 95 = 9025 rainbow tables to cover all possible hash variants. It is already prohibitively expensive for most hackers to calculate rainbow tables for passwords of reasonable length – adding the salt to the mix completely rules this out.

Salting adds entropy to user passwords, making them harder to crack. Some systems may use the same salt on all accounts – this is not a good solution, as an attacker who learns that salt during a breach can add it to their systems to crack the hashes. Good, strong systems generate salt on a per-user basis to make that much harder.

Since a salt value is generated every time a user generates a new password, it's also important to understand that they need to be stored somewhere. The system still needs the original salt used to hash a user's password whenever that user tries to log in, so that it can be re-hashed with their entered password to compare to the hashed version in the password store.

You may be surprised to know that the salt is frequently stored, in plain text, along with the password itself. To see an example of this, launch the CentOS VM, log in, open a terminal window, become the root user, and look at the file **/etc/shadow**.

For the **student** account, you will see this entry in the password field (the last characters aren't shown for reasons of space):

6T5TjncC5Xc37QXYm$sT1VLtrUdIBPqGPN/PBFcOqWGb2VMgYFqs3mLuY.BR6BeuZ6J9dKXWp…

This single password field is actually divided into multiple smaller fields, each delimited by the **$** character. The first field identifies the hashing algorithm used for this password. In this case, it is **6**, which indicates the password was hashed with the SHA-512-based variant of the **crypt** algorithm.

The next field is the salt, which in this case is **T5TjncC5Xc37QXYm** – you can see that this is a long random value. That alone is considerably longer and more random than any user-selected password.

The final (remaining) field is the hash of the salt + the password.

If you're going to try to crack a password with a salted hash, you already know that you can most likely rule out using rainbow tables (since the salt massively increases the number of tables you'd need to generate, to the extent that it would be impossible to store them all), but that doesn't mean that those passwords still can't be cracked. As long as you know the salt itself, you can still attempt regular (brute-force) cracks that will work if the password itself is weak.

15. Let's assume you have found a hashed password for a root user in a Unix system, and you can also see the salt (since it is stored in plain text). You might find that the salt is "**sa**" and that the hash is "**ded9a412c94192044eb6bbca59d11098**".

    The first step is to understand what sort of hashing algorithm has been used. There are a number of tools that can do this. In a web browser, go to https://www.tunnelsup.com/hash-analyzer/ which is one such online tool, and paste the hash above into it.

    What is the hash type? _____

16. So far when using **hashcat**, you've been specifying the command-line option **-m 0** which specifies what type of hashing algorithm you want **hashcat** to use (which in turn is determined by the hashes you have that you're trying to crack). Enter the command:

```
./hashcat-cli64.app -h
```

to see a list of the various hash methods that **hashcat** can use. What is the hash type for mode 0 that you've been using to date?

_____

You might also be interested in running **hashcat** with the **-b** option, which performs a benchmark process, telling you how many hashes of the various types that it supports can be calculated every second on your CPU.

17. This time, you don't just have a plain MD5 hash of a password – instead, you have an MD5 hash of a password plus a salt. Fortunately, **hashcat** can include a salt in its hashing process when it tries to crack the password for you. Look back through those hash methods that **hashcat** showed you in its help output. Which two hash methods does **hashcat** support where the MD5 hash is generated from both a password and a salt?

_____

18. To try to crack your salted password, you need to get your (known) hash and salt into a plain text file. Back in your Macintosh terminal window, using a simple Unix text editor on the Mac (such as **vi** or **nano**), create a file called **crackme.txt** and on the first line, enter the hash (**ded9a412c94192044eb6bbca59d11098** from above), and the salt (**sa**, also from above), separated by a colon. Don't include any other characters like spaces or quotes. Save the file, and then try the following command in the terminal:

```
./hashcat-cli64.app -m hashType -a 0 crackme.txt passwordList.txt
```

Were you able to recover the password for that hash and salt? What is the password?

_____

Which of the two hash methods was able to crack the password?

_____

19. Now try one more variant. This time, you have found a salted hash in a Unix system that looks as follows (it is a single line, but takes two here because it is wider than the page):

```
$6$iqcHZcTC0/sSxkHi$f.jtvRg7QXyB4ueoofNxwBYZad2KsH7j4OmcUPXU3q3hNl67bsUdNMR
QHXBnstj9fzNprBgZFLm2FLKiD4Avw1
```

tunnelsup.com won't reliably recognise this hash and salt combination, but it consists of a salt (**iqcHZcTC0/sSxkHi**) followed by a **sha512-crypt** hash of the salt and password.

hashcat has a method for cracking salted hashes that use this algorithm, and if you look through the hash methods displayed earlier, you might think that it's method **1710** – but it's actually **1800** in this case, and **hashcat** can understand the full line exactly as it is shown above (in other words, you don't need to separate out the salt from the hash).

Re-edit **crackme.txt** and replace the existing hash and salt with the salted hash above.

20. Next, try to crack this salted hash using **passwordList.txt** as your source of passwords:

```
./hashcat-cli64.app -m 1800 -a 0 crackme.txt passwordList.txt
```

21. You should find that you can't crack it – that's because the password isn't in that file. What is the next biggest list of passwords you have that you could try?

_____

Try to crack it using this file instead. (**hashcat** will display a continuation prompt when you run it this time – it is still grinding away in the background, and you can get a status update at any time by pressing the **s** key). It should take you around 3 and a half minutes to crack this salted hash.

What password did you find?

_____

Final Thought: How safe are you?

As a final step for this part of today's tutorial, consider checking your own security profile.

Go to the site https://haveibeenpwned.com and enter your university (or other) email address into the **email address** field. This site is a reputable way to check whether your email address is included in any of a large number of security breaches that have occurred in recent years where the results of the breach have been made public. If your account has been used at a breached site, you may want to ensure you change your password at that site, and anywhere else where you use that email address.

Finally, go to https://haveibeenpwned.com/Passwords at the same site. This is a safe and secure way to check if any of your passwords have been found in any of the security breaches that the site knows about. Again, if any of your passwords are found to be in any breaches, you should take steps to change those passwords and be sure to choose strong passwords when you do so.

Part 2 – Forensics

Digital forensics, as you saw back in lecture 11, is about responding to an incident that has occurred, and collecting the evidence that relates to that incident for investigation. It may be about trying to discover what took place, or it may be about trying to simply recover lost data.

In a recent tutorial you configured two firewalls and an intrusion detection system (IDS). Both of these items of network infrastructure are important lines in network defence, but they can also be rich sources of information for anyone attempting to build an audit trail of events that have occurred on a network. If a breach occurs, these are two sources that you would likely be examining very closely in an attempt to discover where the activity originated, and how the perpetrator managed to get into your system. In today's tutorial you won't be examining this kind of data, but it is important to remember the role that audit logs play in your forensic activity. During the tutorial you will see that audit logs exist stored on host systems as well, and these can be a source of data for learning what has taken place.

Today you will be focusing on host-based forensics, and not the network. When an incident occurs, you need to capture information about the host (or hosts) that are affected. In the forensics lecture this process was summarised as "The Three A's": Acquire, Authenticate, Analyse. This could involve the processes such as hashing disk images, dumping RAM, and collating important files on the host, in addition to the generation of reports about the software that was running on the host when the incident occurred.

One particular activity you will spend a substantial amount of time on today is the recovery of deleted files. There are many tools that enable you to do this - you can use them directly on a host, or you can do it after you have recovered a complete disk image of the target machine.

Activity

22. Launch the Windows 7 VM and open Windows File Explorer and go to the E: drive.

23. Create several text files and folders. Open the files and add some content. Drag some PDFs from the Macintosh Desktop's KIT304 folder into the VM window, and then drag those files into the E: drive as well. Try not to be too random here – **you need to remember what you created.**

24. Now select all of the files in the E: drive, and delete them, and then empty the recycle bin. These are the files that you will now attempt to view from forensic tools.

25. Shutdown the Windows 7 VM, and when it's powered down, close its window.

26. Next, you want to configure the Windows 7 VM to boot from an ISO disc image, rather than the built-in hard drive. In the VMWare Virtual Machine list:

- right click on the Windows 7 VM and choose the **Settings...** option
- click the **Add Device…** button in the upper-right of the window
- click on the **CD/DVD Drive** icon and click the **Add…** button on the lower right
- click on the **Autodetect** menu, and select **Choose a disc or disc image…**
- in the dialog that opens, click the **Desktop** icon (or type **Command-D**), then click on the **KIT304** folder, then click on the **Tutorial 19** folder, and select the **caine6.0.iso** file, and finally, click the **Open** button.
- in the Connect CD/DVD Drive window title, click the **Show All** button, and click the **Startup Disk** icon on the lower left
- click on the **CD/DVD** icon, and close the VM settings window

27. You'll need to be paying attention, and ready at the keyboard for this next step. Start up the Windows 7 VM, and as soon as you see the CD-ROM boot screen, press the down-arrow key several times to select the boot option **Boot Live in debug mode**, then press **enter**. This will boot the Caine Linux distribution that contains a range of different forensic tools.

28. Since this operating system doesn't have the VMWare tools installed, it can't respond to screen size changes that you'd normally manage just be resizing the window. To make the screen larger, click the **Menu** button at the bottom left of the Caine screen and select **System** > **Preferences** > **Monitors**. Now find the **Resolution** setting on the upper right, and choose the 1440x900 value, then click **Apply**, and in the confirmation window, click **Keep this configuration** and finally click the **Close** button.

29. Before you can start forensically exploring your E: drive to attempt to recover the deleted files, you need to mount a hard drive that is writable, so you have somewhere to recover those files to (remember that when carrying out Forensics exercises, you don't want to modify the original media, so the E: drive must remain read-only – hence the need for an additional drive for the recovery operation).

    At the bottom left of the Caine screen, click the **Menu** button and select Accessories > Disks. You'll see that there are three Hard Disks (27GB – the Windows 7 boot volume, 53 MB – the E: drive, and 21GB – a previously unused volume you'll use for the recovery). Click on the 21 GB drive (**/dev/sdc**). In the right-hand panel, you will see a graph of the disk's partitions (there's only the one). Click on the icon under the partition map that looks like two cogs (on mouse over it displays **More actions**) and select **Edit Mount Options**. In the middle of the popup window you'll see a long field of comma-separated values. Delete the first value **ro** and its comma. This option makes the volume mount as **Read Only**, and by deleting it the drive will mount with write access. Click the **OK** button to close the dialog.

30. Now you need to mount this drive. To do so press the "play" button (the sideways triangle) to the left of the cogs button. Once it mounts, you'll see the drive appear in the **Other Devices** section at the bottom of the **Devices** list. You can now close the **Disks** utility, and you will see that there is now a drive on the Desktop called **New Volume**.

31. Next you will attempt to recover the files that you deleted from drive E. The first step is to create an *image* of the drive in question. When conducting forensics after an incident has occurred, you should always ensure that you don't modify any original media.

    There are two common methods for imaging the disk. The first, **dd**, is a Unix command-line tool that has been used since the 1980s to acquire the contents of a hard drive. It produces an exact copy without modifying it. However, it does not support the compression of the acquired image and nor does it record any metadata about the image or its collection. **EnCase** is the other useful option which does support both image compression and metadata recording. It is also referred to as the **Expert Witness Format**. Many tools support both of these formats.

32. On the Caine desktop you will see a tool named **Guymager**. This is one of the more popular forensic image creation tools and it supports both **dd** and **EnCase**. Launch **Guymager** by double-clicking its icon. It will display a list of drives connected to the system. Locate the one connected to **`/dev/sdb`** – this is the 53 MB volume that appears as drive E: on Windows 7. Right-Click on it and select **Acquire image**.

    In the window that opens, leave the **File format** setting on **Expert Witness Format**. You get to play Forensic Police Officer - fill in some details relating to Case and Evidence numbers, the Examiner (you) and provide a Description. In the **Destination** section, enter **`/media/sdc1`** (which is the mount point of the 20G drive you mounted at step 29) as the location to save the image. Give the image a filename, and a related but different name for the info file that relates to this image.

    The **Guymager** image creation tool supports a number of hashing options that can be used to prove the integrity of the image. You can leave this on the default (MD5) setting, but you can see that it also supports SHA-1 and SHA-256. Ensure the checkbox to verify the image is enabled, and click the **Start** button.

33. Due to the very small size of the source drive (50 MB) the imaging process only takes a few seconds to run and then verify. Close the **Guymager** window when it's done.

34. Open the **New Volume** drive on the desktop and (amidst the other Windows files that are present on Windows NTFS volumes, but that Windows itself normally hides from view), you'll see the two files that were just created. One is an image of the drive with the **`.E01`** file extension and the other is the associated information tile with the extension **`.info`**. Open the **`.info`** file and browse its content. Confirm that the details of the MD5 integrity check are present.

35. Next you'll use a tool for conducting file system analysis. This is an interactive web-based GUI called **Autopsy** that is built on an open-source suite of tools called **Sleuth Kit** – there's a description over the page of the kinds of things it can do. To run Autopsy, click the **Menu** button at the bottom left of the Caine screen, and select **Forensic Tools** > **Autopsy 2.24**. It will open a

terminal window (which you need to keep open), a web server hosting the GUI and the Sleuth Kit tools, and then launch the home page in Firefox.

Since Autopsy is a forensics tool, it uses the notion of a "case". You'll need to provide:

- a case name
- the host (or hosts) you're investigating
- the details of the image(s) that were taken from the host(s) that you're investigating

The Autopsy Browser provides the following evidence search functionality:

- **File Listing**: Analyze the files and directories, including the names of deleted files and files with Unicode-based names.
- **File Content**: The contents of files can be viewed in raw, hex, or the ASCII strings can be extracted. When data is interpreted, Autopsy sanitizes it to prevent damage to the local analysis system. Autopsy does not use any client-side scripting languages.
- **Hash Databases**: Lookup unknown files in a hash database to quickly identify it as good or bad. Autopsy uses the NIST National Software Reference Library (NSRL) and user created databases of known good and known bad files.
- **File Type Sorting**: Sort the files based on their internal signatures to identify files of a known type. Autopsy can also extract only graphic images (including thumbnails). The extension of the file will also be compared to the file type to identify files that may have had their extension changed to hide them.
- **Timeline of File Activity**: A timeline of file activity can help identify areas of a file system that may contain evidence. Autopsy can create timelines that contain entries for the Modified, Access, and Change (MAC) times of both allocated and unallocated files.
- **Keyword Search**: Keyword searches of the file system image can be performed using ASCII strings and grep regular expressions. Searches can be performed on either the full file system image or just the unallocated space. An index file can be created for faster searches. Strings that are frequently searched for can be easily configured into Autopsy for automated searching.
- **Meta Data Analysis**: Meta Data structures contain the details about files and directories. Autopsy allows you to view the details of any meta data structure in the file system. This is useful for recovering deleted content. Autopsy will search the directories to identify the full path of the file that has allocated the structure.
- **Data Unit Analysis**: Data Units are where the file content is stored. Autopsy allows you to view the contents of any data unit in a variety of formats including ASCII, hexdump, and strings. The file type is also given and Autopsy will search the meta data structures to identify which has allocated the data unit.
- **Image Details**: File system details can be viewed, including on-disk layout and times of activity. This mode provides information that is useful during data recovery.

Source: http://digital-forensics.sans.org/blog/2009/05/11/a-step-by-step- introduction-to-using-the-autopsy-forensic-browser/

36. To get started with Autopsy, click **New Case**. Enter a case name (which can't contain spaces or other special characters), a description, and your name as an investigator, then click the **New Case** button to create the case. Autopsy will show you where it's created the case files, and will then display the **Add Host** button to go to the next step (click this).

    Next you need to provide the details of the host you're investigating (again, this can't contain spaces or other special characters). You can leave the other fields at their default values, and click the **Add Host** button to proceed to the next screen. Autopsy will show you where it's created the host files, and will then display the **Add Image** button to go to the next step (click this).

    Next, you need to tell Autopsy where the image file(s) are that are part of the case. Click the **Add Image File** button. In the Location field, you need to enter the path to and name of the image file you created back in step 33. The path to the image should be **`/media/sdc1/`*`imagename`*`.E01`**. (If you can't remember the name, minimise the Firefox window, and open the **New Volume** icon to determine the name of the image file – it should be the one with the **`.E01`** extension). Leave the other values on the page at their default values, and click **Next** twice (assuming it successfully located the image).

    Finally, click the **Add** button and then click the **OK** button to see the **Case Gallery**.

37. From the **Case Gallery** you can select a drive for analysis. Select the second drive on the list (which represents the file system on the imaged drive, rather than the entire raw drive), and click the **Analyze** button. In the new screen that's displayed, click the **File Analysis** button at the left-hand end of the row of buttons at the top of the screen.

    This enables you to look through the image and view the files that it contains. You'll also see the deleted files are shown in red text (Note that the first letter of the file names is missing). You should be able to locate the files you deleted, and you should be able to click on deleted folder names and see the files they contain.

38. Click the **All Deleted Files** button on the left-hand side of the window, and you will see a single list on the right of all the files from all the directories on the drive that have been recovered. In the list of files, you can click on any filename to view that file as ASCII or HEX data. You can also export them to a writable filesystem, or create notes about them that are then saved to the Case you created.

39. Click the **Close** button at the right-hand side of the row of buttons across the top of the browser window to return to the drive list, then click the **Close Host** button, and then click the **Close Case** button.

Caine has many other uses, including discovering passwords, root kits, recovering corrupted or virus-infected machines, and detecting steganographic content in images.

To finish you are going to have a quick look at a similar product produced by the same team that develops Caine, but that runs under Windows. This is called Win-UFO (Ultimate Forensics Outflow), and is basically a collection of over 100 different tools all bundled together in one utility.

40. To proceed, you want to get Windows 7 running again. To do this:

    - click the **Menu** button at the bottom left of the Caine screen and select **Shut down**
    - when the OS tells you in the console to **Please remove installation media and close the tray**, press the return key
    - right-click on **Win7** in the VMWare Virtual Machine list, and choose **Settings…**
    - in the window that opens, click on the **CD/DVD** icon, click the **Advanced options** triangle, and then click **Remove CD/DVD Drive**
    - launch Windows 7 as normal by double-clicking it in the virtual machine list

41. Open the Windows Explorer and open the **Win-UFO** folder at the root of the C: drive. Then open the **Reports** folder, and open the **ROOT**… folder you find inside. Double-click the **Win-UFO-Report.html** file. This is the output of an automated analysis of the system by **Win-UFO** that took just under half an hour to complete and has been completed already for you.

    Browse through the file. Note that Win-UFO has created hashes of many files that are important from a forensics perspective: log files, registry files, configuration and network files. It also takes a snap shot (of sorts) of the processes running on the system at the time the report was generated.

42. Now you're going to run a different audit program – **WinAudit** – to see the sort of information it can gather compared to **Win-UFO**.

    - In the Windows file explorer, return to the top-level Win-UFO folder, and launch **Win-UFO** by double-clicking on it
    - click **Yes** when the User Account Control dialog is displayed, and accept the license agreement
    - when prompted if you want to create reports for your investigation, click **No**
    - select the **Reports** tab and then select **WinAudit**
    - click on the **Here** link in the middle of the screen – it will take about 30 seconds for the audit process to finish

    Once the audit is complete, can you find the names of all of the user accounts on the system? There is a significant amount of information in the report, and if you were using this to investigate following an incident, it does provide you with a good summary of what was running on the system when it occurred, what versions of software were running, and so on.

43. Win-UFO contains many different kinds of tools. Move the mouse over the different utilities to see a description of each one. Unfortunately the Windows 7 VM is not particularly interesting as it is continually reset back to a "clean" state when you sign out, and there are no other applications installed onto it that you might want to extract useful information about.

    To illustrate this look at the **Password Recovery** tab, and roll the mouse over each of the utilities inside. These enable a user to extract passwords from many different applications. It also includes tools for capturing passwords as they get sent through the network card, however, we most of those applications aren't installed in the image, and you aren't transmitting any passwords in this tutorial to capture.

44. Open the **Log Viewer** tab, and click on the **LastActivityView** and **RecentFilesView** to see how the activity history of the system is easily accessible. These provide a simple visualisation of the content of some of the log files that you saw the hashes for in the report you saw in step 41.

45. Open the **Recovery** tab and run **Recuva**. This enables you to recover the deleted files on the E: drive, this time in Windows using a GUI interface. Progress through the wizard, choosing to recover all files using the **Deep Scan** option to the Desktop folder on the **C:** drive.

46. In the **Browser History** tab there is a large selection of tools for interacting with many modern browsers. On this machine we only have Internet Explorer and Firefox, each with a fairly limited history, but you can use those tools to view the pages that the virtual machine has visited over time. Can you deduce from these logs when this virtual machine image was first created?

Win-UFO contains tools that enable you to take a "dump" of the current content of RAM on your machine, view EnCase images of hard drives, and scan networks. Feel free to play with these, but in your setup this week there is nothing particularly interesting to discover using these more advanced tools.

Conclusion

In part 1 of this tutorial, you've learned more about passwords, password strength, password cracking, and techniques to strengthen passwords against attacks such rainbow tables through the use of salting.

You should now understand that people are generally very bad at choosing strong passwords that are resistant to cracking attacks, but at the same time, you are now more likely to consider your own password use, and the techniques that you use to choose and to protect them.

In part 2 of this tutorial you have explored a small slice of the large range of forensic software that can be used, and you've seen the "forensic approach" of documenting (with case files) everything that's done – an important aspect of the chain of custody.

Skills to Master

This week's skills that you need to master to progress through the unit include:

- cracking password hashes, both salted and plain, using **hashcat**
- using **rcrack** on Windows 7 to crack passwords using rainbow tables
- Undertaking simple forensic exercises using tools in the Caine distribution.