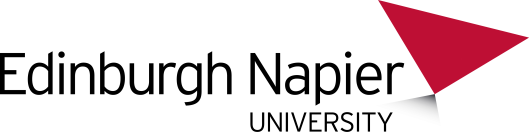
****

**Coursework Assessment Brief**

|  |  |
| --- | --- |
| **1. Module number** | *CSN08114 & CSN08414* |
| **2. Module title** | *Scripting for Cybersecurity and Networks* |
| **3. Module leader** | *Dr Petra Leimich* |
| **4. Tutor with responsibility for this Assessment**  Student’s first point of contact | *As above* |
| **5. Assessment** | *Coursework - Python Script Development - Coursework with Lab Demo* |
| **6. Weighting** | *60% of module assessment* |
| **7. Size and/or time limits for assessment** | *You should be able to complete this assessment within approximately 20 hours (if you have kept up with the Lectures and Labs). Online Demo will be approx. 15 mins explaining the various scripts/demoing selected functionality.* |
| **8. Deadline of submission** | **Monday 6 December 15:00pm.**  Your attention is drawn to the penalties for late submissions.  Only your programme leader can authorise extensions.  **Demo sessions in the week beginning 6/12/21.** |
| **9. Arrangements for submission** | *Submit a zip of your code and other specified files to Moodle by the submission deadline. You must keep your own copies of your assessment deliverables and bring them to the demo (working code and documentation).*  *By submitting to moodle you confirm that the work is your own and that any code used from templates or online sources has been identified as such through inline comments. It is YOUR responsibility to ensure that the online submission has completed successfully.* |
| **10. Assessment Regulations** | All assessments are subject to the University Regulations.  **You must not share your work with other students - this includes posting any of your work in any repository that is accessible to others (such as GitHub) and applies also after you have completed the course**. The student conduct regulations contain a detailed definition of academic misconduct which includes use of commissioned material (which you would be doing if you post questions on stackoverflow, for example); knowingly permitting another student to copy all or part of his/her own work. See <https://my.napier.ac.uk/Student-Administration/Conduct/Pages/Conduct.aspx> for details (follow the Regulations link there). |
| **11. The requirements for the assessment** | See specification overleaf. |
| **12. Special instructions** | During the demo, you will be asked to run your code, and to explain some of the code in detail. Code documentation will also be reviewed.  Demos will be held on campus and/or online depending on the covid situation at the time of the demo. |
| **13. Return of work and feedback** | You must keep a copy of your work.  Individual oral feedback will be given during the demo (you are advised to take notes). A written summary of the feedback with indicative marks will be available via Moodle within three weeks of the demo date (holiday periods excepted).  Please note that all marks are subject to internal moderation and verification at the assessment board. If you have difficulty understanding your feedback or would like to discuss it, you should contact the module leader in the first instance. |
| **14. Assessment criteria** | See specification overleaf. |

# Assessment overview

The analysis of network traces (packet captures) is core to any cybersecurity operation. In this coursework, you will therefore **build a network trace (pcap) analyser utility using Python 3** (3.7 or higher). This will open a packet capture (pcap file) and parse it for specified information such as IP and email addresses. It will also perform some statistical analysis on the contents and visualise the traffic flows.

Your code must be documented appropriately. It must also be PEP8 compliant and get a pylint score as close to 10/10 as possible. Use separate functions for each aspect of the functionality, and group these together appropriately in separate modularised scripts. You will have coded some of the required functionality in lab exercises - and could import those modules rather than copying the code. The requirements are described in detail below.

During the demo, you are required to run your analyser with the pcap file **evidence-packet-analysis.pcap (available in moodle)** as the test case.

# Functionality Requirements - read very carefully! Refer to the separate FAQs for additional explanations.

Points 1-10 below correspond directly to the 10 marking criteria overleaf, while 0 is a general requirement.

1. While your script is running, it should **print status information to the standard output** (screen) as well as the specific output from the requirements listed in the following points. By status information, we mean, for example, "file xxx read successfully", "results written to file yyyy", etc.
2. Open, parse and close a pcap file. For the demo, this will be **evidence-packet-analysis.pcap**, but it should be easy to replace this with a different file. Store the parsed packets in Python, in one or more objects, for later re-use. Choose suitable data types for the object(s).
3. Summarise the contents of the capture in a table format, **with one row for** **each traffic type**.
   * Each row should show the **number of packets of that type**, the **first** and **last** **timestamp**s and the **mean packet length** for that type.
   * The mean packet length should be of the entire packet object (not just the TCP payload). Make use of the in-built len() function instead of the embedded len attribute in the packet.
   * The example packet capture contains only three types (UDP, TCP, IGMP), but for full marks your script should handle the types **adaptively** and add other packet types if they occur. To test this, you should use other pcap files, see additional information below for some suggestions.
4. Find the information listed below. For some of these, you need to use regex, others are fields in the request parts of packets or can be extracted using the os module.
   1. Find any email addresses present in **To**: fields from emails in the packets
   2. Find any email addresses present in **From**: fields from emails included in the packets
   3. Find the **URIs** of all requests for image (e.g., jpg, gif, PNG) files and
   4. extract the **filenames** of the requested image files.

Print the unique email addresses, full URIs and filenames to screen in a suitable format.

1. Extract the **sender and destination IP address pairs** for all packets and count how many packets were sent from/to each.
   * Return the results in the form of a **dictionary**.
   * Consider carefully what you should use as the dictionary key.
   * Print the results to screen **sorted** so that most frequent traffic comes first, and nicely formatted.
2. Find the **geolocation information** for every unique destination IP address for which this is available, and **create a KML** file that can be opened e.g., in Google Earth.
   * Include only the destination IP addresses.
   * For each IP address, you should include city, country, latitude and longitude.
   * You will need to deal with incomplete information – this can be part of your exception handling
   * Store the results in a KML file.   
     Choose a suitable structure.   
     Each point should include the packet count and city (if available) in its information.
3. **Plot** the **number of packets** against time as a **line chart**.   
   Part a. is about the preparation, while part b. is the actual graph. Each part is worth 3 marks.
   1. To start, you need to group packets into equal length time intervals, and store the start time and the number of packets for each interval.
   * Experiment with the length of time interval to get a good graph – it needs to be long enough for each interval to contain a good number of packets and short enough to give you a bit of a timeline.
   * The length should be a variable that is easy to change.
   * Calculate the threshold for exceptionally heavy traffic. A rule of thumb in statistics is to use the mean number of packets per interval plus two standard deviations.
   1. Plot the data as a line chart.
   * It is expected that you use matplotlib.pyplot, but you can choose different tools if you prefer.
   * Make sure the plot is annotated well – using suitable x- and y-axis labels, caption etc.
   * **Show the threshold** visually in your plot. This could be a horizontal line, or represented e.g. by using different colours.
   * Display the plot on screen automatically while the program runs and **save it as a PNG** file.
4. Include **comprehensive exception handling** in your scripts.

* For example, you should check for non-existent files and allow for packet captures with different types of traffic to be parsed and processed successfully.
* Any output from Exception handling should be directed to sys.stdout.
* Test your exception handling with several pcap files, such as the ones used in the lab and some of the ones mentioned in the additional information below.

1. Consider the **overall design, style, readability and efficiency** of your code.
2. Each element of the functionality should be achieved by one (or more) functions. The functions should be grouped appropriately into several separate modules.
3. The **central script** which is run to call the code must be named **pcap\_analyser.py**.
4. Use variables rather than hardcoding, use return statements to assign results to variables for future use.
5. Do not repeat the same operations or code unnecessarily.
6. Document functions, modules etc with appropriate doc strings.
7. Use additional comments in key places to explain complex code.
8. Reference/cite any websites and other sources of code from others that you make use of. Give the specific source (e.g. the stackoverflow page, ideally also the author).
9. Create a dependency diagram using the instructions in moodle and submit it with your code.
10. Make your code **PEP8 and pylint compliant**. Refer to the PEP8 style guide (<https://www.python.org/dev/peps/pep-0008/>). Run your code through pycodestyle and pylint. Submit evidence of this in a file.
11. During the **demo**, you will be asked to run your code. You **must** use the pcap file **evidence-packet-analysis.pcap as the default** test case.  
    You will be asked to explain selected aspects of your code. Make sure you know your code and its structure well.

# Additional Information

* You are allowed to use a suitable library other than dpkt to parse the pcap if you wish, for example scapy. You are allowed to use any standard and third party packages/libraries that you wish, whether or not they were introduced in the teaching materials of the module. If you are using any "unusual" 3rd party packages, include a readme.txt file in your submission which contains clear, concise installation instructions for these non-standard modules. Packages available in PyPi are preferred.
* If you are not familiar with network packet captures at all, have a look e.g. at <https://computer.howstuffworks.com/question525.htm> to understand the basics.
* To check that your script's output is as expected, open the pcap file in Wireshark and inspect it there. You can find an intro to Wireshark e.g. at <https://sweetcode.io/wireshark-packet-analyzing-tool/>. This page also has a link to many more pcap files you can use for testing. <http://forensicscontest.com/2009/09/25/puzzle-1-anns-bad-aim> and <https://asecuritysite.com/forensics/pcap> also have pcap files recommended for testing - choose one or two.
* All code must run on Windows. If you should wish to use functionality that is only available in Linux, this must be agreed with the module leader beforehand.
* This coursework makes use of several components that you have already completed as lab exercises. Refer to the corresponding exercise descriptions on the lab sheets and any related lecture slides for a detailed description of their functionality.
* **Always “future-proof” your code.** Do not hardcode anything that might be changed at a later date and use variables wherever possible. If you do hardcode something (e.g. the name and path of the pcap file or a directory for saving output files), do this only once and only in the main script, so that it is easy to change.
* Keep it simple and build your code bit by bit. Don't worry if you can't get it all working at first.
* Test regex with an online Python regex tester, e.g. <https://pythex.org/>.
* Start working on this coursework as soon as possible. Get help in the labs and read the FAQs.

# Marking

Categories 1-8 are marked on the functionality implemented. You can gain marks for code even if it doesn't run – please read FAQ document for more info. Category 9 assesses your understanding of the code, as shown during the demo.

There are 4 marks available for all categories except 6 and 8, for which there are 6 marks. Half marks may also be given.

|  |  |  |  |
| --- | --- | --- | --- |
| Marked out of 4 | Marked out of 6 | General description for criteria 1-8 | Demo description |
| 4 marks | 6 marks | Excellent, full implementation, making use of functions, return statements etc. | Excellent explanations and answers to questions |
| 3 marks | 4-5 marks | Very good implementation with one or two minor aspects that could be improved. | very good explanations and answers to questions |
| 2 marks | 2.5-3 marks | Good implementation but incomplete or with one or two more major issues. | good explanations and answers to questions |
| 1 mark | 1-2 marks | Insufficient attempt at implementation: not working or very incomplete. | insufficient explanations and answers to questions |
| 0 marks | 0 marks | no attempt visible in submitted code. | No demo |

IF YOU DO NOT ATTEND THE DEMO, YOUR SUBMISSION WILL STILL BE MARKED BUT YOU CAN GET MAX half marks for each of categories 1-8 and 0 for the demo, i.e. max 18 marks or 45% total.

|  |  |  |
| --- | --- | --- |
|  | Functionality/Scripts Required | Max marks |
| 1 | Open and parse a pcap file. Store the contents of the file in one or more python objects for later reuse. | [4 marks] |
| 2 | Output a summary of the packets in a table format, with one row for each packet type. Each row should show the number of packets of that type, the first and last timestamps and the mean packet length for that type. | [4 marks] |
| 3 | Find the email addresses in (a) To: and (b) From: fields of any emails included in the packets. Further, find (c) the URIs of all requests for image (e.g., jpg, gif, PNG) files and (d) extract the filenames themselves.  Print unique email addresses, full URIs and filenames to screen in a suitable format. | [4 marks] |
| 4 | Extract the sender and destination IP address pairs for all packets and count how many packets were sent from/to each address pair. Print the results to screen sorted so that most frequent traffic comes first and nicely formatted. | [4 marks] |
| 5 | Find thegeolocation information for every unique destination IP address for which this is available, and create a KML file that can be opened e.g. in Google Earth. | [4 marks] |
| 6 | Plot the number of packets against time, and flag up any time intervals with exceptionally heavy traffic (e.g. more than the mean plus 2 standard deviations). Save the plot as a PNG file. | [6 marks] |
| 7 | Use comprehensive exception handling throughout your code. | [4 marks] |
| 8 | Modular, future-proof code that uses variables, functions and return statements to facilitate reuse. Good documentation (doc strings, references, comments) and suitable structure. Some consideration for efficiency.  Clear dependency diagram; PEP8 compliant code with a high pylint score. | [6 marks] |
| 9 | Demo: running the test case, explanations of code and answers to related questions. | [4 marks] |
|  | Total | [40] |

# Submission checklist

Zip together the files below and then upload the zip to moodle. DO NOT submit the test pcap file.

|  |  |
| --- | --- |
| item | Included in zip? |
| Main code file, called pcap\_analyser.py |  |
| All other code (.py) files |  |
| Copy of kml file from #5 |  |
| Copy of png file from #6 |  |
| Dependency diagram |  |
| Evidence of pycodestyle and pylint checking |  |
| If you are creating a jupyter notebook as part of your coursework include it in the zip |  |
| If applicable, a readme.txt file with any 3rd party python packages required, and where to find them |  |
| Do not include a copy of the test pcap file |  |