1. Identify a cybersecurity website that contains data that can be scraped and processed via a script.
2. Design and Implement a script, or series of scripts, that can automate the gathering and formatting of the data.
3. Run the script and record as a screencast. There are several free options for screencasting – try Screencast-O-Matic or OBS. If need be, you can use you phone to video the script running. **You must submit both the script and the video screencast.**
4. Write a 500-word report on your solution.

**The report should:**

1. Evaluates the quality, reliability, efficiency and security of the code that you wrote.
2. Identifies the limitations and challenges of your solution, and suggests what you could improve for next time.
3. Describes the testing you have done to ensure the functionality of your script.

Any references used should apply the normal referencing guidelines.

## Cyber scrape and search

The script provides a search function for data downloaded from cyber.gov.au. The cyber security related information on the site is prolific, wide ranging and at times in-depth. This makes the site a valuable resource for learning about cyber security. However, the search feature on the site is poor. The outcome of learning can be applied to job interviews, educational pursuits and on-the-job support through knowledge lookups.

### Quality

The quality of the code is reasonable as measured against aesthetics, useability and functionality. Colors, borders and spacing improve some of the display, though the text editor display is only as good as the text editor chosen by the user. The script is very usable with a simple interface and simple menu options. Functionality is somewhat limited in the area of search as only a very basic grep search option is provided

### Reliability

The script is reliable as all required files are downloaded; the number of downloaded html files always equals the number of converted text documents; the program runs till completion without error; string match frequency is always correctly calculated and the existence of installed prerequisite programs is checked

The script has some unreliability as partial string variations can cause misleading results. For example, the search term ‘attacks’ does not match occurrences of ‘attack’ or ‘attacked’. Also, brackets around a searched phrase in a file will affect the search. For example, a search term of ‘ddos’ will not match occurrences of ‘(ddos)’. In addition, double quotes added to a search term are not filtered and will prevent matching.

### Efficiency

The script is efficient as it runs quickly; user inputs are acted on quickly; repetition loops reduce code use; functions are used to keep code clean, and menu options are minimised. The script has some inefficiencies as the ‘cut’ command does not support change file in place and so additional files are created through redirection. Also, repeated scrapes will download the same files again and in hindsight there may have been no need to convert html to text as the html can be easily searched.

### Security

The script is secure as it holds no confidential information and it filters javascript statements from html files as they are downloaded, thereby potentially preventing download of some types of malicious code

### Improvements, limitations and challenges

The search feature is very basic and consists only of using grep to search for a line matching the search phrase with multiple search strings having to occur adjacent to each other. Capacity to search for partial matches as discussed previously would be valuable, as would boolean logic options and other features provided in traditional search engines such as Google. Other future improvements could include

* downloading more files
* applying the program to any site rather than just cyber.gov.au
* providing a user with a list of potential search terms
* adding more statistics for searches
* providing a better display agent than a text editor, such as lynx html viewer
* searching html rather than converting to text and searching
* Clickable links in output to redirect to webpage

### Testing

|  |  |
| --- | --- |
| **Checks for potential error conditions** | **Error successfully rejected** |
| Attempt to display URL when no string matches were found | 🗹 |
| Attempt to display local files when no string matches were found | 🗹 |
| Enter file access numbers higher than the numbered lines displayed | 🗹 |
| Enter single and multiple search strings | 🗹 |
| Enter strings in double or single quotes | 🗷 |
| Check menu options for rejection of invalid options | 🗹 |
| Attempt to run search before scraped data is available | 🗹 |

|  |  |
| --- | --- |
| **Checks for correct functioning** | **Functions correctly** |
| Local file display includes line numbering | 🗹 |
| Display of duplicate weblinks with matching strings was supressed | 🗹 |
| Number of duplicate weblinks was correctly displayed | 🗹 |
| URL table displayed with correct content, colors, borders and spaces | 🗹 |
| Curl, sed, cut, sort, uniq and nl, provided the correct output | 🗹 |
| Grep matches search string, ignores case and acts recursively | 🗹 |
| Grep, where required, suppressed colors | 🗹 |
| Multiple strings used in search phrase was successful | 🗹 |
| Search strings matched whole and partial words | 🗹 |
| Menu operation | 🗹 |
| Online and downloaded files have the same name | 🗹 |
| Converted text files do not contain strange characters | 🗹 |
| Headers and footers removed from converted text files | 🗹 |
| Number of files in html and text directories is equal | 🗹 |
| Output files contain correctly formatted strings (urls.txt; links2) | 🗹 |
| Range and number of downloaded files is correct | 🗹 |

Figure 1 – The script is ran from the command line with an optional argument for choosing a text editor



Figure 2 – A menu is presented with options for scraping and searching

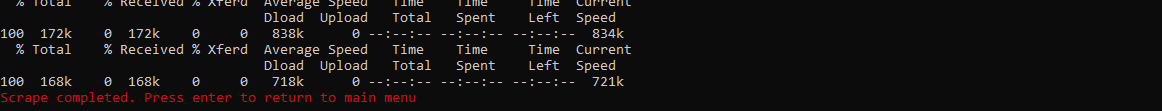
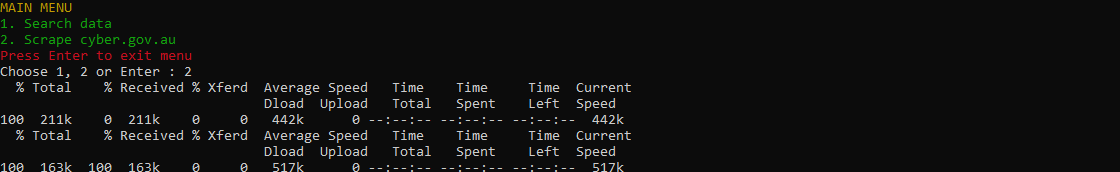


Figure 3 – A scrape takes a minute or two to complete and then a user is notified of completion

Figure 4 – After scraping and choosing the search option, entry of search phrases can begin and the results are displayed showing the matching line and the local file holding the line

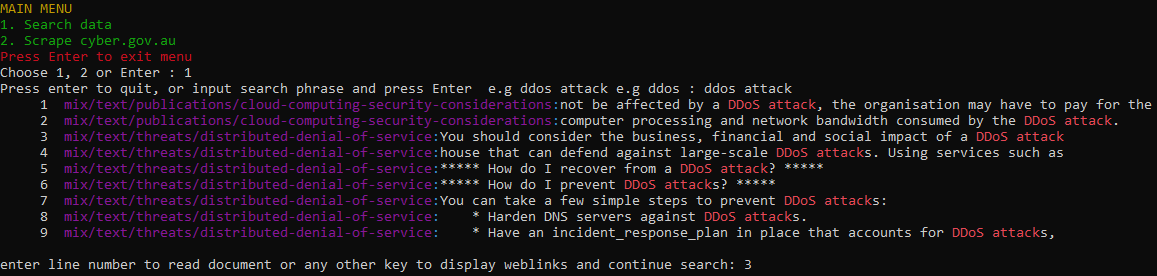


Figure 5 – Choosing the line number on the results page, opens the page in a text viewer/editor, which is Nano by default, or the editor specified as a command-line argument

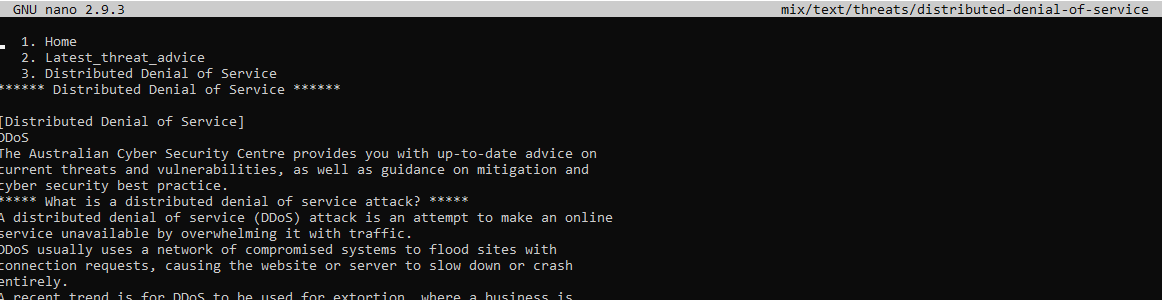


Figure 6 – When finished viewing the file, options are available to view another file, or display the URLs from which the text files were generated, and then return to searching

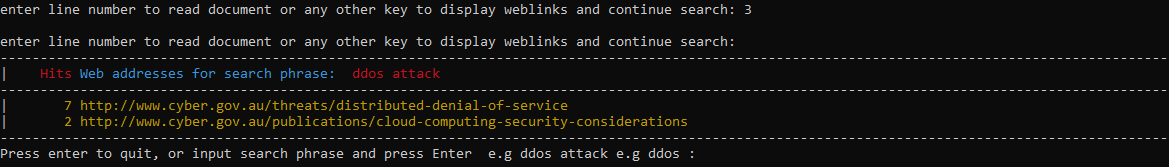


Figure 7 – pressing Enter, exits the program