I. Compare and contrast rule-based detection and anomaly detection using web server CLF log files, in terms of types of attacks they can identify, advantages and limitations.

Two main attack detection techniques exist: rule-based detection and anomaly-based detection. Rule-based detection defines a number of static rules that a log entries must adhere to and are deemed to be an attack if they do not meet the criteria; malicious. In contrast, anomaly-based detection works in two parts: a training phase where the application is fed a file that contains non-malicious entries in order to develop an understanding of normal traffic and then a testing phase where an additional log file is provided and it is checked for malicious traffic.

Rule-based systems operate in a fashion that they attempt to match predefined patterns against entries in a log file; such as regular expressions on a line to check for cross-site scripting or to monitor various system log files for high levels of unauthorised requests and ban those I.P addresses for a defined amount of time, like Fail2Ban [1]. Following this, the environment where rule-based systems are used can then choose to adopt one of two security models: positive (whitelist) where anything matching a rule is allowed, or negative (blacklist) which is the inverse. Simple rule-based systems can be exceptionally effective at detecting known attack types, such as SQL injection, through the use of regular expressions. Through these rules there becomes very clear distinction as to whether or not a request was malicious. Implementing sophisticated rules can be incredibly complex [2] and inline with this attacks are becoming more sophisticated [3] and so too should rule-based detection systems.

Anomaly-based systems build up a profile how how an application or system should look in it's logs through the use of normal activities. What the tool will define as normal behaviour is dependant on the implementation of the tool, however, standard metrics can be an expected range of HTTP requests per hour, geography of those requests or an expected number of unauthorised HTTP requests. Where they differ to rule-based is that they develop a profile of the installed machine and analyse previous traffic to see if any of it is outside of a given range, in contrast to a clear divide between what is or isn't an attack with rule-based systems.

Where anomaly-detection shows it's limitations is in its ability to predict how the system will behave in the coming months. A website that receives a spike in traffic due to advertising and uses anomaly-detection system would result in a false-positive due to the traffic being out of the expected range. This is inherently a problem in general with detection techniques and their lack of an ability to correlate a large number of data sources. This results in a high false alarm rate due to the anomaly-detection algorithms basing their result on past events. In addition to this, the system is told what 'normal' data looks like and not what unusual data looks like. While this does indeed allow the system to report anomalies, it does not allow for the system to detect stealthy attacks and this onus falls on to the developer of the detection system to implement the something that may be able to detect this.

Where both of these detection systems fail is in their ability to detect *new* attack types. While an anomaly-based detection system may be retrained on the previous n-weeks, it will not be able to detect a new attack type and this will go undetected until a systems administrator detects it and updates the tool. In a bid to solve this problem, neural network log file anomaly systems are being researched and have seen good results with lower false-positives in contrast to traditional methods [4][5][6]. These new deep learning methods present their own challenges with evolving log file structures and thus when the application itself is changed, the neural network must be retrained and tested, which naturally can take a very long period of time; especially in contrast to the traditional detection methods. While deep learning methods offer the ability to detect anomalies in evolving and unorganised data structures, it is doubtful that they will replace the existing systems in an enterprise environment until they can effectively run in an unsupervised fashion with an unstructured and evolving log file.

II. Describe how the tool works

To prevent excessive memory consumption while running the tool, it is implemented to stream only one line in the file at a time and to extract only the required information from a given line; the current date and the requested path (URI). While training, this information is extracted and stored in to a `Map` where the key is the URI and it's value is a `Path` object that is used to store information about the requests that were made. Inside this `Path` another `Map` exists that contains a key of the hour the request was made and a value that is the number of requests (hits) that were made. If the current HTTP request was made on a weekday then `Map` entry's key is the hour of the request, if it was made on a weekend then the key is the current hour + 24; this ensures that all of the requests for a `Path` are stored together.

For a given interval `i`, $\bar{x}_i$ is it's mean (1) and $\bar{x}_i$ is it's sample standard deviation (2); where `H` is the number of requests it has received, `N` is the length of the interval and x_i is the number of requests for a given hour.

$$\bar{x}_i = \frac{H}{N} \tag{1}$$

$$s_i = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N}(x_i - \bar{x})^2} \tag{2}$$

The tool contains a method that iterates over every line in a file and uses regular expressions to extract the: IP, date, URI, HTTP status code, and size. This method receives a callback which is used to call either a training or test method and provides it with a line's information. For brevity, this is left out of the pseudocode below.

Algorithm 1: Training

```
for line in training file do
    increment hits for this URI and hour
for Path in paths do
    calculate means;
    calculate standard deviations (Algorithm 3)
```

Algorithm 2: Testing

```
for line in test file do
    if current hour != last then
        for entry in test hit Map do
            if hits is outside of interval then
                print actual and expected values
        clear testing map
    else
        increment test hits for this URI and hour
    set the last date to the current date
```

Algorithm 3: Standard deviation

```
for map entry set do
    boundary[weekend?][intervalId] += variance
for isWeekend.length(2) i do
    for boundaries.length(4) j do
        $boundary[i][j][0] \leftarrow mean - (\alpha * sd)$
        $boundary[i][j][1] \leftarrow mean + (\alpha * sd)$
```

During training lines are streamed and data is stored, only the current hour is considered. While the provided files were small, this ensures that for both testing and training resources are kept to a minimum in the event that large files are provided to the tool; this was tested by duplicating the data within the files to around 10GB each.

To store the means and calculated interval boundaries for a path, multidimensional arrays are used. The mean for an interval is divided by the length of the interval (in hours) multiplied by the number of times that interval was seen; this prevents the mean growing proportionally to the length of the log file. In order to calculate array index access, a method is implemented that checks which interval an hour resides in; `means = new double[2][4], intervalBoundaries = new double[2][4][2]` For example, given the specification, hour 18:00 is in interval 3 and so index 2 is returned. Following this, if this hour was calculated to be a weekend `(hour > 23 ?)`, then index 1 is returned. Using this methodology, the calculated data is arranged in an efficient manner.

Data for the training set is stored in to a map that contains values that are `Path` objects which too contain maps. Both of these maps are `HashMap`s which have `O(n)` space complexity and the time complexity is `O(1)` for both storage and retrieval of the data in these maps. The time complexity for the training algorithm is $O(n)$ for the testing algorithm as the map size never exceeds 48, and $O(n^2)$ for the testing algorithm. This is the best case time complexity given that the file is streamed and total hourly hits are checked against interval boundaries.

III. Discuss the differences between the anomaly detection results obtained using the outlier coefficient values α=1.0, α=2.0, α=3.0, α=4.0 and α=5.0.

To fully understand the results that have been obtained, more information is required about the routes themselves (E.g, is /mailman/listinfo/support a protected route?), business staffing hours and the HTTP request origins. This is because at face value some of the routes that appear to be protected may not be and them experiencing high levels of traffic may be acceptable.

α	**Results**	**Lower**	**Upper**	**Most queried URI:hits**
1	4356	1151	3205	/mailman/listinfo/faculty=588
2	2960	637	2323	//mailman/listinfo/faculty=448
3	1981	297	1684	/mailman/listinfo/faculty=369
4	1409	145	1264	/mailman/listinfo/faculty=292
5	1032	80	952	/mailman/listinfo/faculty=228

TABLE I
RESULTS FOR EACH COEFFICIENT OF α

Table I details the number of results that were returned for each of the values of α. The results here detail an exponential decrease in the number of results. For all the five alpha values, 95 of the suspicious results were as a result of the lower and upper boundary values being the same. This is as a result of the variance in the standard deviation equation being zero due to the number of hourly hits an hour received being equal to the mean. A variation in the statistical analysis methods used would resolve this issue, however, at present this is not an acceptable number. While the path `/mailman/listinfo/faculty` is the most queried path across the test log file, this path also suffers from the issue with the variance being equal to zero, an example: `/mailman/listinfo/faculty 5 [6.20, 6.20]`.

This tool shows a number of weaknesses when using values of α of 1 or 2 in that there is minimal grace in flexibility of the hourly hits that the paths receive. This is demonstrated in a value of 1 and there being 4356 results; a large portion of these are where the number of hourly hits is +/- 1 of the boundary at such a low number of expected hits. For the provided log files and low average hourly hits that are experienced in the environment where the mailing server is used it appears that a higher α value - more standard deviations - would produce less false-positives. The choice of an acceptable α becomes environment specific but also increasing the value lowers the effectiveness of the tool in detecting anomalies. While 95 false positives do exist in the dataset, even without them there still exist a large number of results that were experienced during the testing phase.

It is clear, however, from the test data generated for α=2 that some paths that appear to be private (`"/mailman/private", "/mailman/admin/sswk"`) have received a higher level of hits than expected at hours that be unusual for the application; for example: 00:00, 04:00. Given this coefficient, the tool appears to have correctly identified an unusual amount of traffic to these paths. One noticeable cause for concern was the following:

```
[Tue Oct 30 21:00:00 GMT 2018] /mailman/admin/sswk/ 8 [4.00, 7.30]
[Tue Oct 30 21:00:00 GMT 2018] /mailman/listinfo/support 13 [3.67, 11.33]
[Tue Oct 30 21:00:00 GMT 2018] /mailman/listinfo/faculty 16 [6.85, 10.85]
[Tue Oct 30 21:00:00 GMT 2018] /mailman/listinfo/students 26 [12.17, 18.13]
[Tue Oct 30 22:00:00 GMT 2018] /mailman/admin/sswk/ 3 [0.84, 1.86]
[Tue Oct 30 22:00:00 GMT 2018] /mailman/listinfo/support 5 [1.01, 3.64]
[Tue Oct 30 23:00:00 GMT 2018] /mailman/private/ 4 [0.28, 2.17]
[Wed Oct 31 00:00:00 GMT 2018] /mailman/admin/sswk/ 4 [0.84, 1.86]
[Wed Oct 31 00:00:00 GMT 2018] /mailman/private/ 4 [0.28, 2.17]
```

Given the above sample, when a coefficient of 5 is used a number of the results are missing and the acceptable upper limit is shifted within a couple of hits of what was found in the test file. This further highlights that the implementer of this tool must strongly consider the suitable coefficient value to use in their environment. A misinformed user of the tool may be lead to believe that as the actual value is quite close to the boundaries that this is an acceptable value, when in fact, it is not.

If an attacker had an understanding of the website and the hours where it experiences lower levels of traffic, they would be able to launch an attack in that hour and possibly go undetected by this tool. This in part highlights an issue that the tool is quite primitive in it's implementation and implementing IP address statistics as well as HTTP request type frequency would further aid in detecting attacks. In addition to this, implementing a per-URI α would further aid in detecting attacks. As if certain paths only receive a very low number of hits at night time, for example, then any increase experienced would be immediately reported; increasing the α value here only allows for an increase in the number of hits where it should not. In addition to this, some paths receive a higher fluctuation in their hourly hits in contrast to others; `"/index.html"` may receive more than `"/mailman/private"`. In conclusion, to develop a more sophisticated tool it should be given an understanding of the environment that it is operating in.

IV. Briefly discuss what types of web sites your tool might be suitable for. For what types of web sites is it unlikely to produce useful results?

This type of tool would be best suited to applications that typically receive traffic from one country or in an internal network where conditions are more predictable. As IP detection is not implemented, the tool has no context as to where a request is made from and as such, a malicious user could send a number of requests from an unexpected range and have those requests served.

While this tool can detect deviations in the number of requests made, it cannot detect anomalies in the request body and so it is not suited towards applications that use SOAP; where all parameters are within the request body. The tool has been designed in such a way that it would be easy to further enhance it's capabilities and check for cross site scripting, SQL injection or similar through inspecting the URI fragment itself. At present, this tool is unlikely to produce acceptable results for an application that is largely serving data through a single request type; for example, a blog. This is due to the tool not currently taking in to consideration the request type. The website could be experiencing lower than normal traffic but then a malicious user makes a large number of POST requests and this could bring the hourly rate up to within the boundaries. Implementing a feature that also calculated statistics that included the request type and size would then allow for this anomaly to be detected.

This tool is best suited to non-mission critical environments and environments where the behaviour is expected to deviate; such as an internal mailing system or an intranet. This is due to the statistical methods that are employed within the tool. If the total number of hits that a path receives in an interval is consistent then the variance for that path's interval becomes zero, thus, the standard deviation is zero and then the calculated lower and upper boundaries are the same. As a result of this, any logs that are evaluated using the tool will throw false positives due to the statistical methods. An alternative to allow for this tools functionality to be used in an environment where this is the case would be to change the statistical methods that are employed within the tool.

It is important that while using this tool that the training data is frequently updated within the tool as otherwise the data will become stale. An application where this tool would struggle is a public-facing website. A public-facing website's traffic could suddenly spike if it is shared on a social network or featured in the news, while this would be an anomaly it would not be malicious and would not require security intervention. This tool would also suit an application that is on a public website but requires user login in order to access the private resources as this would highlight any abnormal traffic to both them and the login page itself. In addition to this, this tool is best suited to websites that have a large deviation in the number of hits per hour for each path. If this is not met then the variance in the standard deviation equation may be zero and this will mandate that the number of hourly hits must equal what was calculated during training.

References

[1] "Main page." [Online]. Available: https://www.fail2ban.org/wiki/index.php/Main_Page

[2] A. Ghorbani, W. Lu, and M. Tavallaee, *Network intrusion detection and prevention concepts and techniques*. Springer, 2010.

[3] C. University, "Cyber risk outlook 2018," 2018. [Online]. Available: https://www.jbs.cam.ac.uk/fileadmin/user_upload/research/centres/risk/downloads/crs-cyber-risk-outlook-2018.pdf

[4] W. Li, "Automatic log analysis using machine learning," Nov 2013. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:667650/FULLTEXT01.pdf

[5] T. Yang and V. Agrawal, "Log file anomaly detection." [Online]. Available: https://cs224d.stanford.edu/reports/YangAgrawal.pdf

[6] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS 17*, 2017. [Online]. Available: https://acmccs.github.io/papers/p1285-duA.pdf