The Application of High-Throughput Testing and Design of Experiments to Cyber Systems

Gaurav Simon Gandhi

The University of New South Wales at the Australian Defence Force Academy

The cyber space has been expanding rapidly in recent years, and with this increase comes the necessity for continual improvement in the test and evaluation of cyber systems. Scientific exploration can be used to gain a better understanding of the space, allowing us to better stop or defend against cyber-attacks. However, to date, there appears to be a lack of published research that has been conducted for testing in the field, and of the research reviewed, even less have used a proper experimental methodology, or a systematic analysis that could provide rigorous results. High-Throughput Testing uses combinatorial methods to reduce the number of test cases in an experiment, whilst ensuring a predetermined coverage level. Combined with Design of Experiments, which allows us to determine the effects of each individual input factor in a system, and interactions between factors which often are not intuitive, we can design efficient and effective experiments that allow us to better understand the system under study. These techniques are already being employed extensively by the U.S. military, to improve test and evaluation procedures. This project was conducted to determine the applicability of High-Throughput Testing and Design of Experiments techniques to testing a simple cyber system, whilst following a proper experimental method, so as to provide the framework for future studies into more complex systems.

Contents

I. Introduction 2
II. Aim and Hypothesis 2
III. Project Scope 2
IV. Technique Overview 3
   A. High-Throughput Testing (HTT) 3
   B. Design of Experiments (DOE) 3
V. Phase 1: Research and Testing 3
VI. Phase 2: Screening 6
VII. Phase 3: Modelling 9
VIII. Future Work 10
IX. Conclusion 10
X. Acknowledgements 10

References 11

APPENDICES
Appendix A. HTT Test Plan A1
Appendix B. Screening Analysis A2
Appendix C. Modelling A3

1 PLTOFF, School of Engineering & Information Technology. ZEIT4500/4501
I. Introduction

In recent years, cyber space has been expanding rapidly, and the benefits of technological advancements are all around us. Despite this, an increasingly connected world requires us to continually adapt to change, and also poses an increase in vulnerabilities that can easily be exploited. The high cost of real-world cyber-attacks, as well as the physical and reputational damage they may inflict, are just some of the reasons that make an improvement in cyber test and evaluation imperative. To date, there appears to be limited published research on effective test methodologies in the cyber field, for which reason the technology sufficient to protect systems isn’t being developed at the rate required. There is a lack of proactive development of broad capabilities of cyber defence, with more of a focus on reactive responses to specific threats [1], and this results in an inability to derive consequences in a systematic way [2]. High-Throughput Testing (HTT) is a combinatorial design methodology, used to reduce the total number of test cases to a minimum whilst ensuring a predetermined coverage level [3]. HTT, when used with Design of Experiment (DOE) techniques, allows us to determine the individual effect of input factors in an experiment, as well as the effect of interactions between factors, for a better understanding of the system under study. The purpose of this paper is to illustrate the applicability of HTT and DOE to testing cyber systems, and utilise a proper experimental method, as a basic case for future research and development.

The US Department of Defence (DoD) is already ‘replacing the budget driven test events, combat scenarios and one-factor-at-a-time approach with a statistically rigorous approach to test design using DOE,’” [4]. Despite this research being conducted, all such research conducted by the US DoD is highly classified and unavailable. A successful result from this experiment, and the subsequent implementation of DOE methods by the Australian Defence Force (ADF), could make cyber test and evaluation more effective and efficient.

This paper begins by defining the aim and hypothesis of the experiment, followed by the scope of the project. The Technique Overview section provides a basic overview of the techniques of HTT and DOE, and this is followed by the 3-phase result of this study: Research and Testing, Screening, and Modelling. The paper concludes with a recommendation for future work, and acknowledgements.

II. Aim and Hypothesis

The aim of this study is to design an experiment, where HTT and DOE techniques are applied to a simple cyber system, whilst employing a proper experimental method to ensure the experiment is clear, precise, repeatable and reproducible.

We know that HTT, a method that uses combinatorial mathematics, will successfully reduce the number of test cases, by the method described in the Technique Overview section below. Once the results from testing these limited number of cases are established, they can be processed and analysed using DOE techniques to determine the effect of each input factor to the security and performance of the cyber system. The benefit of such a systemic analysis of the cyber system, is that the areas of weakness, and the input factors that consequently cause them, in the system under study have been identified, allowing us to model systems with a desired level of performance or security.

The objectives in determining a hypothesis statement for this experiment was to have a statement that defines a clear end-state and is also falsifiable. Thus, the hypothesis for this experiment is: High-Throughput Testing and Design of Experiments techniques can be applied to a simple cyber system, whereby aspects of the security and performance of the system can be directly attributed to the significance of one or an interaction of multiple input factors. This statement is falsified if the outputs of the experiment cannot be attributed to the significance of the inputs, indicating a poor experimental model or poor experimental procedure.

III. Project Scope

A. Research and Testing

In this phase, all the preliminary research was conducted, with a study of DOE and HTT methods and techniques, as well as conventional cyber test and evaluation procedures. Further research was required in the cyber space, in order to effectively model a cyber system for testing.

rdExpert™ was used to conduct the High-Throughput Testing of the system, and thus reduce the number of test cases.

The test cases output by rdExpert™ were tested in a virtual machine environment, and the results for security and performance were recorded for further analysis.
B. Screening

In this phase, the results from rdExpert™ are analysed using Quantum XL software, which uses DOE techniques, and consequently the input factors to the system that are were not significant to each of the outputs were screened out of the experiment. Using DOE techniques, we were also able to determine which input factors were the most important for each of the outputs.

C. Modelling

In this phase, Quantum XL is used to create an optimum model of the system, based on both the results obtained in the Research and Testing phase, and the statistical analysis conducted in the screening phase.

IV. Technique Overview

A. High-Throughput Testing

High Throughput Testing (HTT) is a fairly new technique which uses combinatorial methods to provide a solution to the problem of excessive test cases and enables a reduction of the total number of test cases whilst ensuring a predetermined coverage level [3, 5]. The technique determines ‘all pair’ test coverage for combinations of many factors [6]. A typical system, with a large number of input factors at various level (configurations of each factor), would require many hundreds and potentially even thousands of combinations if each combination was to be studied.

The term all pair test coverage, or pairwise testing as opposed to conventional all combination test coverage, is based on the premise that all possible pairs of parameter values are covered by at least one test [5]. Most failures in a system are caused due to the interaction of a few variables, so we design a number of tests, whereby all pairs of input factors are tested in this smaller number of tests.

Orthogonal Array Testing adds one more condition, and that is that all possible pairwise combinations between input factors occur only once, and each vector in the orthogonal array matrix is statistically independent of the others.

rdExpert™ is a software package that can conduct HTT and allows the choice of producing one of two design options: Orthogonal Array (OA) and Pairwise Value Ordering (PAVO). The benefit of producing an Orthogonal Array is that we are able to estimate the effects of certain input factors independently. Pairwise Value Ordering generates a nearly orthogonal design, but has other advantages including three-way combinations in the test design, and generating fewer test runs than OA. Despite this, in order to determine the effect of each individual effect, OA was selected.

B. Design of Experiments

Design of Experiments techniques are able to provide orthogonal estimates of each input factor, information about the interaction of factors, as well as the way the total system works, something which is not obtainable through conventional one-factor-at-a-time testing (OFAT) [7]. The techniques within DOE fit response data to statistical mathematics, which allows us to better understand the system, create an effective model of the system, and then optimize the input factors to find the best combination of values [8]. Traditional OFAT methods are time consuming, costly, and inefficient, which can thus result in a high risk of error, and the interactions between factors cannot be analysed.

DOE techniques can be run on a number of software packages including Quantum XL, and while the US DoD is already including these techniques in their test and evaluation procedures, the ADF is only starting to realize the significance of the techniques now [9]. As described in the project scope, the experiment, once prepared, is conducted, and the results analysed on software. From here the analysis is incorporated in creating a model of the system, and in the real world the results of the data analysis of the model will be fed back into the input factor selection step, for further iterations of DOE techniques.

V. Phase 1: Research and Testing

The usual process for beginning an experiment which implements HTT and DOE techniques is to first develop a process flow diagram for the system under study, then conduct a cause and effect (CE) analysis to ensure the capture of the causal elements that influence the effect on the system under study, and finally create a DOE model of a system, which depicts the input factors or independent variables for the experiment, the outputs or dependent variables, as well as the uncontrollable factors (noise) and the controlled variables. This can be thought of as a bottom-up approach to the study.

A bottom-up approach in DOE is appropriate for many systems, however assessing the security of a cyber system can be a never-ending task, as if we dig deeper and deeper into the system, more vulnerabilities will inevitably be exposed, paired with new methods of exploitation. For any cyber study, clearly specifying the purpose of the study, and the resources available, keeps the level of effort for the study in check [10].
For this reason, the purpose of the experiment, and resources available were clearly specified. The purpose of the study, as described in the hypothesis, is to show that HTT and DOE can be applied to a cyber system. For this reason, the cyber system designed is as simple as possible, providing the framework for future study into more complex systems.

The approach used for this experiment can be thought of as a top-down approach, whereby the DOE model of the system is created based on what is available. This simplification allowed the project to be completed on schedule, but the expectation remained that potentially poor results could be attributed back to the lack of a CE analysis, and thus the leaving out of important independent variables. The DOE model created is presented as Fig. 1.

The cyber system model that was tested is a single web server architecture. As seen in Fig. 1, all the factors to the left of the system are input factors, or the independent variables for the experiment, and all the factors to the right of the system are the output factors, or the dependent variables. While not depicted, typical DOE models also include factors below the system as well, representing noise factors, or factors that aren’t being controlled that may vary the dependent variables.

**A: Inputs (Independent Variables)**

The Operating Systems chosen to be emulated were Ubuntu, CentOS, Linux Mint and Debian. The decision to keep the systems within the Linux environment was chosen to simplify the experiment and lessen the time for configuration changes between tests.

The Base Memory represents the RAM allocated to the machines. Initially, the RAM chosen for testing included larger values, however as the testing was completed in the virtual machine environment on a laptop,
higher values of RAM allocated to single machines significantly impaired the operation of the laptop, hence the values were adjusted.

A dynamic load imposed on the system simulates a certain number of concurrent users accessing the website. To simulate this, the tool Siege was used, which is an open-source load testing utility, widely used for the performance and load testing of web applications. We can select a number of concurrent users; in our case 25, 50 and 75, to access a website for a certain period of time; in our case 5 minutes. The tool then returns an output which includes average response time, and this time is recorded as an output.

The type of firewall used for this experiment was packet filtering, which monitors outgoing and incoming packets. The two levels of operation were Walled and Open: in the Walled setting, only the ports necessary for http, https and the munin-node (80, 443, and 4949/tcp) were kept open, and the remaining ports were walled.

ClamAV is an open-source antivirus software toolkit used to detect malicious software and viruses [11] and is the software that has been used in this experiment. ClamAV is easy to use and customizable, which was ideal for the purpose of this study. While generally an on-demand scanner, the installation of the ClamAV daemon allowed it to function as an on-access scanner, such that the antivirus operated whilst the testing was conducted.

Mandatory Access Control is a system-wide policy which decrees who is allowed to have access to elements of the system [12]. AppArmor is a Linux Security Module implementation of name-based access control, and is automatically installed on Ubuntu and Linux Mint. AppArmor was also installed on Debian. SELinux can also enforce mandatory access control, and is automatically installed on CentOS.

The Web Server Application input defines the type of the website being served by the web server: a WordPress blog, a Vanilla Forum or a MediaWiki page.

Finally, the network structure variable represents the connection between the server and the client machine, and the 2 levels for this were chosen as host-only network and internal network. The difference between these options is that the host-only network creates a virtual network interface on the host, providing connectivity among the virtual machines and the host, while the internal network is visible to selected virtual machines but not the host [13].

B: Outputs (Dependent Variables)

The dependent variables are on the right of the box, and these are observable and measurable variables that vary due to intervention caused by the independent variable. The dependent variables have been divided into a security aspect, and a performance aspect. The security of the web server machine is measured by a vulnerability analysis, performed by the tool Nessus. Vulnerability analysis involves discovering a subset of input space with which a malicious user can exploit logical errors in a system to drive it into an insecure state [14]. Such an analysis of the security posture of the system is vital; if the breach points/loopholes are found by an attacker, they can lead to data loss and fraudulent intrusion activities [15]. The security of the system is a quantitative output, known as the Common Vulnerability Scoring System (CVSS). CVSS is standardised vulnerability score across the industry (recorded future). The score given to any single vulnerability ranges from zero to 10, and this gives a general idea of how easy it is to exploit the vulnerability, and how damaging it might be. This scoring system is derived from the National Vulnerability Database (NVD) and was chosen as an industry recognized method of comparing the vulnerability of different systems. For the purpose of this experiment, Nessus outputs the vulnerabilities present on the web server, and the CVSS scores for each individual vulnerability are added to produce one total score.

The performance of the system is divided into 5 outputs: Server Response, CPU Usage (User), CPU Usage (System), Memory Usage (Applications) and Load Average. These outputs are measured using Munin, which is a free and open-source computer system monitoring tool. The client machine is known as the Munin Master, and plugins are installed on the web server, known as the Munin Node, which allow the client machine to monitor the performance of the web server. The inclusion of performance as an output of the experiment is important, as it allows us to create a model of a system that balances security and performance, rather than producing a system that is very secure with low performance, or vice versa.

C: HTT and Testing

The next step in the process is to input the factors and their levels into rdExpert™, to reduce the number of test cases required. The test plan produced by rdExpert™ is presented as Appendix A. HTT selected 32 tests out of a possible 2304 for a full factorial test.

From here, testing was conducted in the virtual machine environment of VirtualBox. A disciplined method was followed for each test. Upon the set-up of the system for each test, a vulnerability scan was conducted from the client machine to the web server, after which a dynamic load was imposed onto the web server from the client machine, and the performance of the server was monitored. Each test was repeated thrice. A further five of the tests were conducted on a second computer, in order to complete a Measurement System Analysis for each output.
VI. Phase 2: Screening

The screening phase involved inputting the results obtained into Quantum XL, to determine which input factors most significantly affected each output factor. The screening phase was divided into a number of sub-phases: Orthogonality Check, Residual Analysis, Regression Analysis and Measurement System Analysis (MSA). The results of the screening phase, the MSA, and an analysis of the results, are presented as Appendix B.

A: Orthogonality Check

Orthogonality allows us to estimate the effects of the input factors independently. To observe the orthogonality, we populate the model of the system with random numbers and observe the variance inflation factor (VIF) of each input. Using random numbers allows the determination of the orthogonality of the model itself, without using any of the obtained data. The VIF quantifies how much the variance is inflated and detects multicollinearity in the model. A value of one for the VIF indicates that the factor is purely orthogonal, and thus there is no multicollinearity. Any value higher than one indicates a relationship with the other factors. We expect the model to be orthogonal as we selected the orthogonal array option for the HTT. The result of the orthogonality check for the output of Memory Usage (Applications) is presented below as Fig. 2, and the results are identical for all of the outputs.

![Figure 2. Orthogonality Check Memory Usage (Applications)](image_url)

We can see that a number of the factors are purely orthogonal, however some of the inputs lack some independence in the testing, namely Operating System and Web Server Application. This level of orthogonality is not a cause for concern, however, further testing to produce more reliable results would ensure the VIF for all the input factors would approach 1.

B: Residual Analysis

Next, a Residual Analysis allows us to determine how well the model of the system produced by the software fits the actual data obtained during testing. If the plots obtained display unwanted patterns, we cannot trust the regression coefficients and other numeric results produced by the software [16]. Two types of residual plots were produced, the first of which was Predicted Output vs. Studentized Residual. For this plot, we would like the scatter to appear as random as possible. The residual is the difference between the predicted and actual value, and a randomly scattered plot indicates that the deterministic portion of the model is quite good at predicting the response; thus only the inherent randomness of any real-world phenomenon remains leftover as error [17]. Any non-random pattern indicates that the deterministic portion of the model is not capturing some information that is influencing the residual values. The second plot is Predicted Output vs. Actual Output, and ideally, we would like all the points to lie on the line y=x, indicating that all the predicted values are equal to the actual values.

For each dependent variable, two models are created, the Y-Hat and the S-Hat. The Y-Hat indicates the output of the system, whereas the S-Hat indicates the variation between results, and this is based on the difference in the values for the three repetitions conducted for each test. Appendix B presents the residual plots for the Y-Hat of each output. As an example, observing Fig. 3, we see that for the Server Response output, the Predicted vs. Studentized Residual plot (top left) is obviously not random, and there is a bias acting on the system for this output. Potentially, a factor that is affecting the server response output hasn’t been accounted for, or more testing will need to be conducted to omit the outliers. The Server Response Y-Hat model is not a strong or reliable model. On the contrary, observing Memory Usage (Applications), we can see a clearly random pattern for the Predicted vs. Studentized Residual plot (bottom left), and the Predicted Output vs. Actual Output (bottom left) plot very
closely follows \( y=x \), indicating that the values predicted by the model of the system are very close to actual values obtained from the experiment. The Memory Usage \( Y \)-Hat model is a strong model.

**C: Regression Analysis**

Using Regression Analysis, we obtain an equation that relates the dependent variable as a function of the independent variables. This analysis allows us to understand the impact of changes in the input values on the output, to predict performance and optimise the design.

Upon entering in the data obtained from testing, factors are screened out in stages, beginning with the quadratic elements, based upon the p value of the factor. If the p value is less than 0.05, the factor is statistically significant, it is shown in red and hence cannot be removed. If it is between 0.05 and 0.10, the term is moderately significant, shown in blue, and these are also kept. If the p value is greater than this, the term is considered insignificant, and the factors removed. It is also important to follow the rule of heredity, such that if a quadratic factor is determined to be statistically significant, the individual factor cannot be removed. For example, as shown in Fig. 3, if the factor \( BB \) is statistically significant, we cannot remove the factor \( B \), even if it isn’t.

Due to the fact that a number of the inputs in this experiment are in categorical form, as opposed to being continuous variables like Dynamic Load and Base Memory, it was necessary to work around using these variables using dummy coding, which is automatically performed by the software. Dummy coding is a way of incorporating nominal variables into regression analysis, turning categories into something a regression can treat as having a binary value of 1 or 0 [18].

Once the significant factors were screened out, the value of \( R^2 \) for our model was observable. \( R^2 \) is the proportion of variation in the data explained by the model and is hence a measure of the strength of the regression model. From here, the Main Effects and Pareto Plots were also produced, which present the most significant independent variables for each dependent variable, and finally a regression equation was produced using the
regression coefficients. The final numerical values for the Memory Usage output is presented as Fig. 4 below, and the Main Effects, Pareto Plots, and regression equations for each output can be found in Appendix B.

Figure 4. Final Regression for Memory Usage (Applications)

We can see that the $R^2$ for the Y-Hat model of Memory Usage is 0.9485, which means that 94.85% of the variation in the data can be explained by the model. Using the coefficients obtained from this final regression, we can now write an equation of the system for both Y-Hat and S-Hat, with the uncoded equation for Y-Hat presented below.

$$\hat{Y} = 291.442 + \left\{ -17.644\text{CentOS} \right\} - 23.158\text{Debian} - 136.387\text{DynamicLoad} + 51.017\text{BaseMemory} + 28.749\text{FirewallOpen} -$$

$$26.039\text{Antivirus} - 12.399\text{MAC} + \left\{ -22.756\text{Vanilla} \right\} + 34.947\text{DynamicLoad}^2 + 37.144\text{BaseMemory}^2$$

The Main Effects plot presents the impact of changing each independent variable on the outputs, and the Pareto Plots are bar graphs, in decreasing order, the most significant input factors for each output. The Main Effects plot for the Y-Hat model of Memory Usage is presented as Fig. 5. We can see that the Memory Usage increases with Base Memory (factor C), but the power of the DOE analysis can be seen by observing the minute changes in memory usage due to the Firewall Setting, Antivirus and Mandatory Access Control. We see that the Memory Usage is slightly higher when the Antivirus is Protecting (setting 1) and the Mandatory Access Control is controlling (setting 1). A more interesting result is that the Memory Usage is slightly higher when the Firewall Setting is Open (setting 2).

We notice that the $R^2$ value is poor for the S-Hat models for all the outputs. This occurs due to the low number of test values provided to create the S-Hat model. Whilst the Y-Hat models have 96 test points (32 tests with 3 repetitions), the S-Hat models have only 32 (measuring the biggest difference between the 3 tests.) Fig. 4 presents the $R^2$ for the S-Hat model as 0.4043, meaning that 40.43% of the variation in the data can be explained by the model. This is not a strong model, and thus we cannot trust the results at this stage. More tests would allow for stronger S-Hat models.

D: Measurement System Analysis

A Measurement System Analysis (MSA) is a tool used to identify and quantify the different sources of variation that affect a measurement system. The MSA conducted for this experiment is a variables-data MSA, whereby the data analyzed is measured on a continuous scale. Such an MSA allows us to assess how much variation is associated with the measurement system, and compare it to the total process variation [19]. Ideally,
we want our variation contribution from both repeatability and reproducibility to be as low as possible. For this experiment, five tests were conducted on a different computer, and the results compared to the original tests to produce an MSA. As a rule of thumb, Total Gage R&R (GRR), the sum of the variation due to repeatability and reproducibility, should be less than 30%. The results of the MSA are presented in Appendix B.

The MSA produces 4 graphs, from which we can assess the measurement system. Considering Output 2: Server Response, we see that the variation accounted for by repeatability is 1.17%, which is very good, and the variation due to reproducibility is 5.03%, which is also very good. Assessing the graphs, we see that the Components of Variation graph, top right, shows that the majority of variation in the results is part to part, or between the tests, which is the desirable outcome. We want variability in the data to come from us varying the independent variables between tests, as opposed to variability in repeating the same test on one computer, or on both.

CVSS, Server Response, Memory Usage and Load Average have acceptable values for the Total Gage R&R. However, this is not the case for Output 3 and 4, CPU Usage User and System. The extremely large variation in the data due to reproducibility for both outputs indicates that whilst the data may be appropriate for use in this study, the results may not be able to be reproduced accurately by someone else.

VII. Phase 3: Modelling

Post regression analysis, once we have determined the input factors that are the most significant for each of the outputs, we can optimise the system to meet certain requirements or constraint. For each output, we can specify lower and upper specification limits. As an example, we can set the lower limit of Output 5: Memory Usage as 400MB, and an upper limit of 500MB to ensure the memory usage isn’t so high as to impair the performance of the machine. Imposing specification limits allows us to constrain the model of the system by the quality measure of defects per million (DPM). Upon the creation of a bell curve for each output using its mean and standard deviation, the defects per million is a measure of how many results for the output will lie outside of the specification limits. Ideally, the goal is to have no DPM. This study explored optimising for three possible states:

**State 1:** Model for minimum Defects Per Million (DPM) (Memory Usage)
**State 2:** Model for minimum DPM (Memory Usage), Firewall, Antivirus and MAC must all be on
**State 3:** Model for minimum DPM (Memory Usage) and minimum CVSS score, Firewall, Antivirus and MAC must all be on

The results of the optimisation for each state are presented as Appendix C. For the optimisation of each state, the software selects a continuous value for the Base Memory and Dynamic Load, and a categorical value for the rest of the independent variables, such that that combination of variables meets the requirement imposed; in this case minimum DPM for memory usage. The optimisation was run twice, both providing a different combination of variables, that result in a value of no DPM for Memory Usage. As we can see in the Capability Analysis presented as Fig. 6 below, this means that no output values will lie outside of the specification limits.

This model created is just a simple example of what can potentially be done in the real world. State 3 imposes a constraint on both an aspect of performance and security, giving us the combination of variables for the lowest
possible DPM. As we can see in Appendix C, a DPM of zero cannot be achieved now, and the lowest achieve value for DPM is 2595.

VIII. Future Work

With more time and resources permitted, this study would have benefitted from a validation phase. Quantum XL outputs a test plan to validate the models created in the Modelling phase. This is something that can be conducted in the future.

There are also a few areas in which further research and development can be conducted to improve this study. Firstly, further work is required on the security aspect of this project. While the implementation of the CVSS score was an appropriate way to assess the security of the system in this study, it was found that changing the security elements of the system, including firewall, antivirus and MAC had no significant impact on the security score. The CVSS method was chosen for this project in line with the resources available and the purpose of the project. Potentially, the same system used in this study can be enhanced by implementing more independent variables, and then obtaining actual vulnerability exploit data by having a red team attempting to exploit the web server. Such a study will move the results of this study closer to real world cyber security research.

Secondly, as discussed, more data is required to produce a good variation model for each of the outputs. Without enough data, we are unable to impose constraints on the variation of the data because of the weakness of these models. To improve the models, a larger number of tests will be required.

Thirdly, further research will need to be conducted into the reproducibility of this study. In order to implement this study into a class lab to be taught in the future, it must be ensured that an experiment performed by a multitude of students on a large number of computers will produce valid results. Potentially, hardware, for example the computer processor used, will need to be included as an independent variable in the study.

Finally, future work can and should be done to apply the techniques of this study to more complex, real world cyber systems. It has been shown that the techniques of HTT and DOE can be applied to cyber systems, and if applied can make cyber experimentation much more efficient and effective.

IX. Conclusions

This project aims to determine the applicability of High-Throughput Testing and Design of Experiments techniques to a simple cyber system, and optimize the cyber resilience, in order to establish the framework for further study in the field. These techniques have been employed in a number of different industries around the world, and the cyber space is one area in which it could be particularly useful, but there has been limited published research into its application. This project was divided into three phases, Research and Testing, Screening and Modelling. The testing and analysis were conducted on a single web server architecture, and it has been shown that the techniques of High-Throughput Testing and Design of Experiments can be applied to cyber systems, thus confirming the hypothesis statement. The successful result of this research warrants further research and implementation by the ADF, to create cyber test and evaluation methodologies that are more efficient and effective.

X. Acknowledgements

I would like to thank my supervisors, Dr. Keith Joiner and FLTLT Kate Yaxley for guiding me through this valuable project. The techniques I have learnt throughout the completion of this project are truly powerful, and I feel fortunate to have been supervised by two experts in the field. I would also like to thank my friends and family for supporting me through the long hours it took to complete this project.
References


## Appendix A – HTT Test Plan

The Application of High-Throughput Testing and Design of Experiments to Cyber Systems

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Ubuntu</td>
<td>25</td>
<td>512</td>
<td>Walled</td>
<td>Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Ubuntu</td>
<td>50</td>
<td>808</td>
<td>Walled</td>
<td>Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Ubuntu</td>
<td>75</td>
<td>1024</td>
<td>Walled</td>
<td>Protected</td>
<td>Controlled</td>
<td>Media Wiki</td>
<td>Internal Network</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Ubuntu</td>
<td>25</td>
<td>1532</td>
<td>Walled</td>
<td>Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Ubuntu</td>
<td>25</td>
<td>512</td>
<td>Open</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>Vanilla Forum</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Ubuntu</td>
<td>50</td>
<td>808</td>
<td>Open</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Ubuntu</td>
<td>75</td>
<td>1024</td>
<td>Open</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Ubuntu</td>
<td>25</td>
<td>1532</td>
<td>Open</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>Media Wiki</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>CentOS</td>
<td>50</td>
<td>512</td>
<td>Walled</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>Media Wiki</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>CentOS</td>
<td>25</td>
<td>808</td>
<td>Walled</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>CentOS</td>
<td>25</td>
<td>1024</td>
<td>Walled</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>CentOS</td>
<td>75</td>
<td>1532</td>
<td>Walled</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>Vanilla Forum</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>CentOS</td>
<td>50</td>
<td>512</td>
<td>Open</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>CentOS</td>
<td>25</td>
<td>808</td>
<td>Open</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>Media Wiki</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>CentOS</td>
<td>25</td>
<td>1024</td>
<td>Open</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>Vanilla Forum</td>
<td>Internal Network</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>CentOS</td>
<td>75</td>
<td>1532</td>
<td>Open</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>Linux Mint</td>
<td>25</td>
<td>512</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>Linux Mint</td>
<td>75</td>
<td>808</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>Vanilla Forum</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>Linux Mint</td>
<td>50</td>
<td>1024</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>Media Wiki</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>Linux Mint</td>
<td>25</td>
<td>1532</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>Linux Mint</td>
<td>25</td>
<td>512</td>
<td>Open</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>Vanilla Forum</td>
<td>Internal Network</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>Linux Mint</td>
<td>75</td>
<td>808</td>
<td>Open</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>Linux Mint</td>
<td>50</td>
<td>1024</td>
<td>Open</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>Linux Mint</td>
<td>25</td>
<td>1532</td>
<td>Open</td>
<td>Protected</td>
<td>Un-controlled</td>
<td>Media Wiki</td>
<td>Internal Network</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>Debian</td>
<td>75</td>
<td>512</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>Media Wiki</td>
<td>Internal Network</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>Debian</td>
<td>25</td>
<td>808</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>Debian</td>
<td>25</td>
<td>1024</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>WordPress</td>
<td>Internal Network</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>Debian</td>
<td>50</td>
<td>1532</td>
<td>Walled</td>
<td>Not Protected</td>
<td>Un-controlled</td>
<td>Vanilla Forum</td>
<td>Internal Network</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>Debian</td>
<td>75</td>
<td>512</td>
<td>Open</td>
<td>Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>Debian</td>
<td>25</td>
<td>808</td>
<td>Open</td>
<td>Protected</td>
<td>Controlled</td>
<td>Vanilla Forum</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>Debian</td>
<td>25</td>
<td>1024</td>
<td>Open</td>
<td>Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Debian</td>
<td>50</td>
<td>1532</td>
<td>Open</td>
<td>Protected</td>
<td>Controlled</td>
<td>WordPress</td>
<td>Host-Only Network</td>
</tr>
</tbody>
</table>
Appendix B – Screening Analysis
The Application of High-Throughput Testing and Design of Experiments to Cyber Systems

Output 1: CVSS (Y-Hat $R^2 = 0.9401$, S-Hat $R^2 = 0.1935$)

\[
\hat{Y} = 23.008 + \left\{ \begin{array}{c}
1.1125CentOS \\
5.4625LinuxMint \\
-8.2875Debian
\end{array} \right\} - 0.63\text{BaseMemory} + \left\{ \begin{array}{c}
-2.0333Vanilla \\
-2.1833MediaWiki
\end{array} \right\}
\]

Measurement System Analysis (Repeatability 0.00%, Reproducibility 0.00%, Total Gage R&R 0.00%)

Screening Analysis - CVSS

The $R^2$ values for the output of CVSS suggests that the Y-Hat model is quite a strong model of the system, whilst the S-Hat model is not as strong, suggesting that potentially an important factor that affects the S-Hat model has been left out, or more data points need to be tested. Observing the residual plots for the Y-Hat model, we observe that the Predicted Y vs. Studentised Residual scatter is quite random, indicating that there is no common trend in the residual values. The Actual Y vs. Predicted Y scatter follows the trend of y=x, indicating that the actual values obtained from experimentation are very close to the values predicted by the model of the system produced by the software.

Experimentation for this output was interesting, as between repetitions on a single computer, as well as between computers for the MSA testing, the CVSS score did not change, which was not desirable. Varying certain security elements of the system, including firewall, antivirus and MAC did not change the CVSS score, which is also not desirable, as we wanted to observe a difference in this output for different security postures of the system. This observation has been depicted in the main effects plots for the S-Hat; all the values of S-Hat are zero.

Observing the main effects plot for Y-Hat, we can see the effect of changing every input variable on the CVSS score.

Operating System (O/S): The CVSS score is greatly varied by the choice of O/S. The score was highest when the O/S is a Linux Mint operating system, and lowest when the it is Debian. Of the four O/S, Linux Mint is the least used, which could be a reason why there is a higher presence of vulnerabilities on this system that haven’t been patched. Debian is a much
more customisable O/S than the other choices, and most elements of the system are not operational until deliberately switched on by the user. This is in comparison to more user-friendly systems such as Ubuntu, in which the O/S is available with many packages and elements already operational, but this results in a larger number of vulnerabilities present.

*Dynamic Load:* Doesn’t affect the CVSS.

*Base Memory:* An increase in base memory results in a linear decrease in CVSS, indicating that some of the vulnerabilities present on the smaller memory machines were present due to the size of the machine.

*Firewall, Antivirus and MAC:* Doesn’t affect the CVSS.

*Web Server Application:* The results indicate that the CVSS score for the service of a WordPress blog is higher than the other two web servers. This result indicates that there are a number of vulnerabilities associated with serving a WordPress blog, and it may be an idea to consider other blog services if security is a primary concern.

*Internal Network Structure:* Doesn’t affect the CVSS

The **Pareto Plots** present the most significant input factors for the output of CVSS, and we can see that the most significant factor is A-4 (Debian O/S) followed by a number of other factors.

The equation presented is the **regression equation** for this output, with uncoded coefficients. The equation is the final form of the model of the system produced by the software. For the continuous variables, such as base memory, we can simply input the actual value of the base memory. For the categorical variables, such as operating system, dummy coding has been implemented to only select one of the four options. If the operating system used was Linux Mint, we would input a 0 as CentOS and Debian, and a 1 as Linux Mint. Ubuntu does not appear in the equation, and if we were to use that operating system, we would simply input a 0 for all CentOS, Linux Mint and Debian. This equation is used in the modelling of the system; by setting constraints on the system, the software predicts a certain combination of input values, and the equation calculates the CVSS.

Finally, the **Measurement System Analysis (MSA)** reflects the lack of variation in the result with multiple repetitions and reproductions. The result of 0.00% variation due to repeatability, and 0.00% reproducibility seems like a perfect result, but in the context of the purpose of the experiment, was not a good result as we weren’t able to compare the security of systems with different security postures. Future studies will need to develop the system to be tested, and/or the experimental procedure and scoring system, to be able to better assess security.
Output 2: Server Response (\(Y\)-Hat \(R^2 = 0.5404\), S-Hat \(R^2 = 0.4597\))

\[\hat{Y} = 1.2557 + \begin{cases} 3.1708 & \text{CentOS} \\ -1.4504L & \text{Linux Mint} \\ -0.3708 & \text{Debian} \end{cases} + 10.136D - 6.7866B - 0.0827F - 0.1748M \]

\[\hat{Y} = -1.6392V - 1.8328W + 2.2895D^2 + 1.1204B^2\]

**NOTE:**
Measurement System Analysis (Repeatability 1.17%, Reproducibility 5.03%, Total Gage R&R 6.21%)

Screening Analysis – Server Response

The $R^2$ values for the output of Server Response initially suggest that both the $Y$-Hat and $S$-Hat models are not very strong, for one of a number of potential reasons. Observing the residual plots for the $Y$-Hat model, we observe that the Predicted $Y$ vs. Studentised Residual scatter has some clear outliers, but the remaining data also fits a certain trend, which is not good. This indicates that the residual value, which is the actual value obtained from experimentation minus the predicted value, tends to decrease linearly as the predicted value increases. This poor residual analysis indicates that the model for this output might be non-linear, and cannot be fit to a linear regression model, or there is a bias in the system that has not been accounted for. The Actual $Y$ vs. Predicted $Y$ scatter does not follow the trend of $y=x$, and again shows that there are some clear outliers in the system. This output requires retesting to begin with, and if similar results are obtained, a different model for the system needs to be considered.

Observing the main effects plot for $Y$-Hat, we can see the effect of changing every input variable on the Server Response score. The trends observed may still be analysed and considered, but the results cannot be interpreted with much accuracy, due to the poor residual results presented earlier.

Operating System (O/S): The Server Response score changes quite a bit with different O/S, and it seems as if running a CentOS operating system tends to result in a higher Server Response value.

Dynamic Load: The dynamic load depicts an interesting trend, such that as the load increases, the Server Response increases at first, and then decreases. Intuitively, as the
number of concurrent users accessing the web page increases, we would expect the Server Response to increase, but this is not seen.

**Base Memory:** An increase in base memory results in a decrease in the Server Response for the system, which is intuitive. However, moving from state 4, which is the higher Base Memory, seems to have a higher Server Response.

**Firewall, Antivirus and MAC:** Firewall and MAC affect the Server Response very slightly. Firewall setting 2 (open) has a higher Server Response, which is not intuitive, however the MAC has a higher Server Response when it is in the ‘controlling’ state, which is intuitive.

**Web Server Application:** The results indicate that the Server Response is lowest when the web page being served is a Vanilla forum web page, and highest when it is a MediaWiki page.

**Internal Network Structure:** Doesn’t affect the CVSS

The **Pareto Plots** present the most significant input factors for the output of Server Response, and we can see that the most significant factor is A-2 (CentOS O/S).

The equation presented is the **regression equation** for this output, with uncoded coefficients. The equation is the final form of the model of the system produced by the software. For the continuous variables, such as base memory, we can simply input the actual value of the base memory. For the categorical variables, such as operating system, dummy coding has been implemented to only select one of the four options. If the operating system used was Linux Mint, we would input a 0 as CentOS and Debian, and a 1 as Linux Mint. Ubuntu does not appear in the equation, and if we were to use that operating system, we would simply input a 0 for all CentOS, Linux Mint and Debian. This equation is used in the modelling of the system; by setting constraints on the system, the software predicts a certain combination of input values, and the equation calculates the Server Response.

Finally, the **Measurement System Analysis (MSA)** for Server Response presented a good result. The variation due to repeatability was 1.17%, which indicates that between the three repetitions for a test, the value obtained did not vary much at all. The variation due to reproducibility was 5.03%, indicating that this experiment could be repeated on another computer, and similar results could be obtained. From the Components of Variation chart presented, we see that most of the variation in the data comes from part to part changes, or between tests due to the changing of the independent variables, which is a good result.
Output 3: CPU Usage User (Y-Hat $R^2 = 0.7125$, S-Hat $R^2 = 0.3261$)

\[
\bar{Y} = 35.325 + \left(\frac{-17.422\text{CentOS}}{9.983\text{LinuxMint}}\right) + 4.04\text{DynamicLoad} + 2.7689\text{BaseMemory} - 4.168\text{FirewallOpen} - \\
4.6695\text{Antivirus} - 4.0232\text{MAC} + \left\{\frac{-1.6392\text{Vanilla}}{1.8328\text{MediaWiki}}\right\} + 8.6293\text{NetworkStructure}
\]

Measurement System Analysis (Repeatability 27.12%, Reproducibility 57.55%, Total Gage R&R 84.67%)

**Screening Analysis – CPU Usage User**

The CPU Usage (User) output is a measure of how much CPU is being used on the processor to run the users code, or code present in the libraries present on the system.

The R² values for the output of Server Response initially suggest that the Y-Hat model is not strong, but is still acceptable, but the S-Hat model is quite weak. Observing the residual plots for the Y-Hat model, we observe that the Predicted Y vs. Studentised Residual scatter is quite random, which is a good result. The Actual Y vs. Predicted Y scatter follows a trend towards y=x, however there is a large amount of scatter, and a few outliers which may have negatively affected the R² result. For future testing, the outliers in the system will need to be retested. If the trend is still not strong after removing outliers, this means that a factor that is important in the variation of data for the CPU Usage has not been included.

Observing the main effects plot for Y-Hat, we can see the effect of changing every input variable on the CPU Usage User score.

*Operating System (O/S):* The most significant result for this input is the minimum point, which indicates that the CPU Usage User value for a CentOS operating system is much lower than the other three operating systems. The Y-Hat values are in percentages; moving from Ubuntu to CentOS there is almost a 30% decrease in CPU Usage User, which is very significant.

*Dynamic Load:* As the dynamic load increases, the CPU Usage User also increases linearly, which is an intuitive result.
**Base Memory:** An increase in base memory results in an increase in the CPU Usage User, which is an interesting result. This suggests that as the base memory of the machine increases, the percentage of CPU Usage by user processes increases.

**Firewall, Antivirus and MAC:** The Firewall input depicts an interesting result; the CPU Usage User is higher when the firewall is not operating. The Antivirus and MAC inputs depict intuitive results, such that the CPU Usage User is higher when they are operating. Any non-intuitive results, such as the result for the firewall, warrant further study into the result.

**Web Server Application:** The CPU Usage User was slightly lower when a MediaWiki page was being served.

**Internal Network Structure:** The result presented for this input was not at all obvious. The internal network structure was the connection between the client machine and the web server, and the results indicate that the CPU Usage User is between 10-20% higher when the connection between the machines is a host only network.

The Pareto Plots present the most significant input factors for the output of CPU Usage User, and we can see that the most significant factor is A-2 (CentOS O/S).

The equation presented is the regression equation for this output, with uncoded coefficients. The equation is the final form of the model of the system produced by the software. For the continuous variables, such as base memory, we can simply input the actual value of the base memory. For the categorical variables, such as operating system, dummy coding has been implemented to only select one of the four options. If the operating system used was Linux Mint, we would input a 0 as CentOS and Debian, and a 1 as Linux Mint. Ubuntu does not appear in the equation, and if we were to use that operating system, we would simply input a 0 for all CentOS, Linux Mint and Debian. This equation is used in the modelling of the system; by setting constraints on the system, the software predicts a certain combination of input values, and the equation calculates the CPU Usage User.

Finally, the Measurement System Analysis (MSA) was not an acceptable result. The variation in the data due to repeatability is 27.12%, and reproducibility is 57.55%, indicating that the results for CPU Usage User still acceptable to observe some trends for repeatability, however the results cannot be reproduced on another computer. By observing the Operator by Part plot, we see that the results of Operator 1 (Computer 1) are always lower than Operator 2 (Computer 2). This indicates that there is a factor that is missing from the analysis, that is crucial to analysing the CPU Usage, and that factor is likely to be something to do with the actual performance of the machine the experiment is being performed on. We also see that between Operator 1 and 2, most of the results for the part numbers are close, however Part Number 2 has a large difference. This part will require some retesting in the future, to finalise the MSA for this output. The results of this MSA indicate that if this experiment were to be performed in a lab environment with multiple students, the results for this output would not be reliable, without prior investigation into the factor that is missing.
Output 4: CPU Usage System (Y-Hat $R^2 = 0.463$, S-Hat $R^2 = 0.5097$)

$$
\hat{Y} = 17.59 + \begin{cases} 
6.5381\text{CentOS} \\
-3.7681\text{Linux Mint} \\
0.7369\text{Debian}
\end{cases} + 19.731\text{Dynamic Load} - 11.771\text{Base Memory} + 1.6419\text{Antivirus} + \\
\begin{cases} 
-3.3633\text{Vanilla} \\
1.9392\text{MediaWiki}
\end{cases} - 4.1715\text{Dynamic Load}^2 + 1.7267\text{Base Memory}^2
$$

Measurement System Analysis (Repeatability 17.89%, Reproducibility 82.11%, Total Gage R&R 100%)

Screening Analysis – CPU Usage System

The CPU Usage (System) output is a measure of how much CPU is being used on the processor to run kernel processes.

The R² values for the output of Server Response initially suggest that both the Y-Hat and S-Hat models are quite weak. Observing the residual plots for the Y-Hat model, we observe that the Predicted Y vs. Studentised Residual scatter doesn’t appear to be random; there is a slight trend of increasing residual values as the predicted values increase. This is known as a heteroskedastic model, such that the dispersion of the error changes over the range of observation. The Actual Y vs. Predicted Y scatter also does not follow the desired trend, and there appear to be plenty of outlier points. These outliers require retesting, but also due to the weakness of this model, a cause and effect analysis will need to be conducted for this output, to make sure no important factors have been left out.

Observing the main effects plot for Y-Hat, we can see the effect of changing every input variable on the CPU Usage System score.

Operating System (O/S): An interesting result is presented here, such that the CPU Usage System is highest for CentOS. In comparison to CPU Usage User, we can see that CentOS uses a greater percentage of the CPU executing system processes, as opposed to user processes.

Dynamic Load: As the dynamic load increases, the CPU Usage System increases, and then begins to decrease at higher values of dynamic load.
**Base Memory:** An increase in base memory results in a decrease in CPU Usage System, which is an intuitive result, and opposite to the result obtained for CPU Usage User. This indicates that as the base memory of a machine increases, a higher percentage of CPU Usage is due to user processes, but this is because the percentage due to system processes is decreasing, not because the number of user processes is increasing.

**Firewall, Antivirus and MAC:** Firewall does not affect the CPU Usage System, indicating that all the firewall operation comes under the category of user processes. When the Antivirus is in ‘protecting’ mode, the CPU Usage System is lower, than when the antivirus is switched off. This is in contrast to the result for CPU Usage User, which suggested that the CPU Usage User is much higher when the antivirus is protecting. This indicates that when the antivirus is operating, it operates as a user process, and hence the percentage of CPU Usage due to user processes is higher, with a corresponding decrease in usage due to system processes. The MAC does not affect the CPU Usage System.

**Web Server Application:** The CPU Usage System is lower when the web server is serving a Vanilla Forum page. This is similar to the result of CPU Usage User, which showed that the CPU Usage due to user processes was lower when serving a Vanilla Forum page, compared to a WordPress blog, and MediaWiki was even lower. We observe that the CPU Usage for Vanilla Forum is lower for both user and system processes; this is an ideal result.

**Internal Network Structure:** Doesn’t affect the CPU Usage System

The **Pareto Plots** present the most significant input factors for the output of CPU Usage System, and we can see that the most significant factor is A-2; operating a CentOS system.

The equation presented is the **regression equation** for this output, with uncoded coefficients. The equation is the final form of the model of the system produced by the software. For the continuous variables, such as base memory, we can simply input the actual value of the base memory. For the categorical variables, such as operating system, dummy coding has been implemented to only select one of the four options. If the operating system used was Linux Mint, we would input a 0 as CentOS and Debian, and a 1 as Linux Mint. Ubuntu does not appear in the equation, and if we were to use that operating system, we would simply input a 0 for all CentOS, Linux Mint and Debian. This equation is used in the modelling of the system; by setting constraints on the system, the software predicts a certain combination of input values, and the equation calculates the CPU Usage System.

Finally, the **Measurement System Analysis (MSA)** was quite poor. The variation in the data due to repeatability is 17.89%, and reproducibility is 82.11%, indicating that the results for CPU Usage User still acceptable to observe some trends for repeatability, however the results cannot be reproduced on another computer. By observing the Operator by Part plot, we see that Parts 1, 4 and 5 have results that are quite close, however Parts 2 and 3 will require some retesting, and potentially a cause and effect analysis will need to be conducted to assess if there are factors missing which are affecting the results. Assessing the Components of Variation chart, we also see that there is almost no Part to Part variation (variation in the data due to the changing of independent variables), and this is a very poor result. The results of this MSA indicate that if this experiment were to be performed in a lab environment with multiple students, the results for this output would not be reliable, without prior investigation into what is causing the difference between operators.
Output 5: Memory Usage (Y-Hat $R^2 = 0.9485$, S-Hat $R^2 = 0.4043$)

\[
\hat{Y} = 291.442 + \begin{cases} 
-17.644 \text{CentOS} \\
21.476 \text{Linux Mint} \\
-23.158 \text{Debian} 
\end{cases} + 136.387 \text{Dynamic Load} + 51.017 \text{Base Memory} + 28.749 \text{Firewall Open} - \\
26.039 \text{Antivirus} + 12.399 \text{MAC} + \begin{cases} 
-22.756 \text{Vanilla} \\
19.994 \text{MediaWiki} 
\end{cases} + 34.947 \text{Dynamic Load}^2 + 37.144 \text{Base Memory}^2
\]

Measurement System Analysis (Repeatability 4.12%, Reproducibility 21.70%, Total Gage R&R 25.82%)

**Screening Analysis – Memory Usage Applications**

This output measured the Memory Usage by Applications on the machine that is being tested.

The $R^2$ values for the output of Memory Usage initially suggest that the Y-Hat model is very strong, and the S-Hat model is not as strong. The S-Hat model would benefit from more test points to create a stronger model.

Observing the **residual plots** for the Y-Hat model, we observe that the Predicted Y vs. Studentised Residual scatter is quite random, which is a good result. The Actual Y vs. Predicted Y scatter follows $y=x$, indicating that close to all of the predicted values are very close to the actual values obtained from experimentation.

Observing the **main effects plot** for Y-Hat, we can see the effect of changing every input variable on the Memory Usage score.

*Operating System (O/S):* The result here seems to show that there is a small effect of changing the O/S on the variation of the data, but this is quite small.

*Dynamic Load:* As the dynamic load increases, the Memory Usage also remains constant; a trend is displayed that shows the Memory Usage decreasing at first and then increasing, however this trend is quite insignificant.
**Base Memory:** An increase in base memory results in an increasing polynomial for the Memory Usage. This result is intuitive; if the base memory of the machine increases, the Memory Usage of the machine will increase, and DOE has confirmed this result.

**Firewall, Antivirus and MAC:** The result for firewall is not intuitive, such that the Memory Usage is higher when the firewall is not operating. The result for Antivirus and MAC show that the Memory Usage is higher when these elements are operating.

**Web Server Application:** The web server application has a minute effect on the Memory Usage results but shows that the Memory Usage is slightly lower when a Vanilla Forum page is being served.

**Internal Network Structure:** Doesn’t affect the Memory Usage

The **Pareto Plots** present the most significant input factors for the output of Memory Usage, and we can see that the most significant factor is C; the base memory, and this factor is much more significant compared to all other factors.

The equation presented is the **regression equation** for this output, with uncoded coefficients. The equation is the final form of the model of the system produced by the software. For the continuous variables, such as base memory, we can simply input the actual value of the base memory. For the categorical variables, such as operating system, dummy coding has been implemented to only select one of the four options. If the operating system used was Linux Mint, we would input a 0 as CentOS and Debian, and a 1 as Linux Mint. Ubuntu does not appear in the equation, and if we were to use that operating system, we would simply input a 0 for all CentOS, Linux Mint and Debian. This equation is used in the modelling of the system; by setting constraints on the system, the software predicts a certain combination of input values, and the equation calculates the Memory Usage.

Finally, the **Measurement System Analysis (MSA)** presents an acceptable result. The variation due to repeatability is very good, at 4.12%. but the variation due to reproducibility is high, at 21.70%. The Total Gage R&R is still acceptable as it is less than 30%. Observing the plot of Operator by Part, we see that all the data points are quite close, except for Part 5, therefore it is highly likely that retesting for this part would produce a change in the results to be a good MSA. After a reassessment of the MSA, in a lab environment, it is expected that this experiment will be able to be reproduced by multiple operators, with similar results.
Output 6: Load Average (Y-Hat $R^2 = 0.9251$, S-Hat $R^2 = 0.1624$)

\[
\hat{Y} = -4.0617 + 2.52291\text{CentOS} + 16.188\text{DynamicLoad} + 4.8307\text{BaseMemory} + 0.9686\text{FirewallOpen} - 0.9003\text{Antivirus} - 1.1826\text{MAC} + \left\{\begin{array}{l}
5.0555\text{Vanilla} - 1.7328\text{MediaWiki}\end{array}\right\} + 1.7264\text{NetworkStructure} - 1.2157\text{BaseMemory}^2
\]

Measurement System Analysis (Repeatability 3.80%, Reproducibility 4.85%, Total Gage R&R 8.65%)

Screening Analysis – Load Average

The $R^2$ values for the output of Load Average initially suggest that the Y-Hat model is very strong, and the S-Hat model is not as strong. The S-Hat model would benefit from more test points to create a stronger model.

Observing the residual plots for the Y-Hat model, we observe that the Predicted Y vs. Studentised Residual scatter is quite random, which is a good result. The Actual Y vs. Predicted Y scatter follows $y=x$, indicating that close to all of the predicted values are very close to the actual values obtained from experimentation.

Observing the main effects plot for Y-Hat, we can see the effect of changing every input variable on the Load Average result.

Operating System (O/S): The result here shows a clear relationship between the operating system and the Load Average. The result shows that when the web server is operating a Debian O/S, the Load Average Y-Hat is much lower compared to the other operating systems.

Dynamic Load: As the dynamic load increases, the Load Average value increases linearly, and this is an intuitive result. As the number of concurrent users accessing the web page increases, the Load Average on the system also increases.

Base Memory: An increase in base memory results in a slight increase at first, followed by a decrease in the value of the Load Average result.
Firewall, Antivirus and MAC: The result for firewall is not intuitive, such that the Load Average is higher when the firewall is not operating. Despite this, the result for Antivirus and MAC show that the Memory Usage is higher when these elements are operating.

Web Server Application: The web server application has a significant effect on the Load Average. The results show that the Load Average for serving a WordPress blog and a MediaWiki page are about the same, however serving a Vanilla Forum has a much higher Load Average.

Internal Network Structure: This result is quite intuitive and shows that when the connection between the client machine and the web server machine is a host only network, the Load Average of the web server is slightly higher.

The Pareto Plots present the most significant input factors for the output of Load Average, and we can see that the most significant factor is B; the dynamic load, and this factor is much more significant compared to all other factors.

The equation presented is the regression equation for this output, with uncoded coefficients. The equation is the final form of the model of the system produced by the software. For the continuous variables, such as base memory, we can simply input the actual value of the base memory. For the categorical variables, such as operating system, dummy coding has been implemented to only select one of the four options. If the operating system used was Linux Mint, we would input a 0 as CentOS and Debian, and a 1 as Linux Mint. Ubuntu does not appear in the equation, and if we were to use that operating system, we would simply input a 0 for all CentOS, Linux Mint and Debian. This equation is used in the modelling of the system; by setting constraints on the system, the software predicts a certain combination of input values, and the equation calculates the Load Average.

Finally, the result of the Measurement System Analysis (MSA) for this output is very good. The variation in the data due to repeatability is 3.80%, and the variation due to reproducibility is 4.85%. Further testing with more data points would improve this MSA even more. By observing the Components of Variation plot, we see that most of the variation for this input comes from Part to Part variation; this is a good result.
Optimisation States: (Memory Usage Specification Limit Range 400-500MB)

State 1: Model for minimum Defects Per Million (DPM) (Memory Usage)

State 2: Model for minimum DPM (Memory Usage), Firewall, Antivirus and MAC must all be on

State 3: Model for minimum DPM (Memory Usage) and minimum CVSS score, Firewall, Antivirus and MAC must all be on

State 1: Optimisation 1

State 1: Optimisation 2

State 1: Capability Analysis
State 2: Optimisation 1

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S-hat</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>PI Lower</td>
<td>25.57</td>
<td>PI Lower</td>
<td>PI Lower</td>
<td>PI Lower</td>
<td>PI Upper</td>
</tr>
<tr>
<td>PI Upper</td>
<td>25.57</td>
<td>PI Upper</td>
<td>PI Upper</td>
<td>PI Upper</td>
<td>PI Upper</td>
</tr>
<tr>
<td>Lower Limit</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
</tr>
<tr>
<td>Upper Limit</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
</tr>
<tr>
<td>Cpk</td>
<td>0.0396</td>
<td>0.0488</td>
<td>0.0488</td>
<td>0.0488</td>
<td>0.0488</td>
</tr>
<tr>
<td>C-p</td>
<td>1.1087</td>
<td>1.0914</td>
<td>1.0914</td>
<td>1.0914</td>
<td>1.0914</td>
</tr>
<tr>
<td>Cp</td>
<td>1.1087</td>
<td>1.0914</td>
<td>1.0914</td>
<td>1.0914</td>
<td>1.0914</td>
</tr>
<tr>
<td>OHP estimate</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>OHP estimate</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

State 2: Capability Analysis

![Probability Distribution Diagram](image)
State 3: Optimisation 1

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Server Response Time</th>
<th>CPU Usage (User)</th>
<th>CPU Usage (System)</th>
<th>Memory Usage</th>
<th>Load Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y-Hat</td>
<td>11.29</td>
<td>3.573</td>
<td>41.113</td>
<td>23.744</td>
<td>7.493</td>
</tr>
<tr>
<td>S-Hat</td>
<td>0.0</td>
<td>0.228</td>
<td>4.0977</td>
<td>2.9927</td>
<td>0.156</td>
</tr>
<tr>
<td>PI Lower</td>
<td>11.25</td>
<td>3.587</td>
<td>41.043</td>
<td>23.716</td>
<td>7.459</td>
</tr>
<tr>
<td>PI Upper</td>
<td>11.25</td>
<td>3.587</td>
<td>41.043</td>
<td>23.716</td>
<td>7.459</td>
</tr>
<tr>
<td>USL</td>
<td>15.0</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
<td>USL</td>
</tr>
<tr>
<td>LSL</td>
<td>0.0</td>
<td>LSL</td>
<td>LSL</td>
<td>LSL</td>
<td>LSL</td>
</tr>
<tr>
<td>Cpu</td>
<td>0.0</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Cpu</td>
<td>0.0</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>DPM estimate</td>
<td>6.780</td>
<td>DPM estimate</td>
<td>DPM estimate</td>
<td>DPM estimate</td>
<td>DPM estimate</td>
</tr>
</tbody>
</table>

State 3: Capability Analysis

![Probability Distribution](image)

- LSL = 400.0
- USL = 500.0