2-18-2016

Detection of malicious content in JSON structured data using multiple concurrent anomaly detection methods

Brett N. Miller

Detection of Malicious Content in JSON Structured Data Using Multiple Concurrent Anomaly Detection Methods

by

Brett N. Miller

Dissertation

Submitted to the College of Technology

Eastern Michigan University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Technology

Concentration: Information Assurance

Dissertation Committee:

Samir Tout, Ph.D. Dissertation Chair

Alphonso Bellamy, Ph.D.

John C. Dugger, Ph.D.

Huei Lee, Ph.D.

February 18th, 2016

Ypsilanti, Michigan

Acknowledgements

There are a number of people I would like to thank for their assistance in this research. First of all, I would like to thank my advisor and dissertation committee chair, Dr. Samir Tout. Dr. Tout's guidance has always been appreciated and has been instrumental in selecting and carrying out this research. I would also like to thank the other members of my dissertation committee, Dr. Alphonso Bellamy, Dr. John Dugger, and Dr. Huei Lee. They have all provided help in numerous ways both in and outside of the dissertation process. I would also like to thank Ted Hanss and the University of Michigan Medical School for their support and encouragement throughout my studies. Thanks also go to my family for their patience and inspiration. And finally, there are too many other people to name who have helped me along the way. It may have been a kind word or some critical bit of information that was useful to me at just the right time. I appreciate them all and only regret that I cannot name them since there were so many.

Abstract

Web applications and Web services often use a data format known as JavaScript Object Notation (JSON) to exchange information. An attacker can tamper with these exchanges to cause the Web service or application to malfunction in a way that is detrimental to the interests of the owners of the Web application or service. Many such applications or services are involved in processes critical to safety or are vital to business interests. Unfortunately, such critical applications cannot always be relied upon to validate the data sent to them. This creates a need for protection external to the applications themselves. This need has been addressed by researchers in other contexts, but there has been little specific focus on JSON and the use of multiple concurrent anomaly detection methods. Some previously proposed solutions involved the detection of known signatures of attacks, but this reduces the chance that new attacks will be recognized. To increase the ability to detect newly created attacks, this research focuses on anomaly detection using general characteristics, rather than the recognition of specific attacks. The detection method this research employs is the Random Forest ensemble algorithm. Metrics such as Shannon entropy, n-gram analysis, JSON structure similarity, character string length, and JSON attribute values are utilized. A goal of this research was the detection of attacks at a rate at least better than chance expectation. This goal was met and exceeded as experimental results using simulated attacks showed considerably better performance. Furthermore, a mathematical model of the interaction of classifier configuration parameters was developed.

Keywords: Intrusion Detection, Anomaly Detection, JSON, Random Forest, Web Application Security, Web Service Security, Shannon Entropy, N-gram analysis

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Web applications and Web services have evolved greatly since their early beginnings in 1992 (Berners-Lee, Cailliau, Groff, & Pollermann, 2010). Web applications are programs that respond when a user visits a Web page or when a program fetches information from a Web service. Web services, which are similar to Web applications, are typically used for system to system communication. In this research, both will often be referred to generically as a “Web application.” Web servers can run one or more Web applications and Web services. Web applications are used in commerce, finance, health care, education, entertainment, as well as Supervisory Control and Data Acquisition (SCADA) systems, which are industrial automation networks. Consequently, a successful attack on such applications and services can have significant impact on society, economic welfare, health, and safety.

Someone determined to attack a Web application may send malicious data to it or tamper with a legitimate stream of data to cause the application to malfunction or to otherwise work against the interests of its owner. Web applications that do not force users to authenticate are particularly prone to attacks since they will accept data from anyone. Tampering can also occur due to the interception and modification of the data stream without detection, particularly if the stream of data between legitimate users and Web application servers is not encrypted. This is because the establishment mechanism of encrypted data streams and subsequent transfers used in Web applications has mechanisms built in to detect impersonation or tampering (Dierks & Rescorla, 2008). Unfortunately, malicious content embedded in data sent to a Web application can be hard to detect. This content poses a “data validation” problem in that a Web application must be able to differentiate between normal and otherwise incorrect, harmful, or malicious data. According to Li and Xue (2014), of the top 10 most serious and prevalent web application attacks listed by the Open Web Application Security Project (OWASP), the top two attacks can be considered to be input validation attacks.

In both Web applications and Web services, data is often sent as packets of structured data. For the purposes of this research, "structured data" will refer to data expressed as letters, numbers, and punctuation, which are part of a structure that has a more complex organization than a simple list of name-value pairs. Extensible Markup Language (XML) and JavaScript Object Notation (JSON) are two common examples of such structured data. XML was created in 1996 as a simpler alternative to Standard Generalized Markup Language (SGML), which was used in the publishing industry (Gray, 2005). According to Smith (2015), Douglas Crockford is credited for the creation of JSON, though Crockford states others also had the idea, but that he provided a name for it and a standardized format. JSON is a text format description of JavaScript data structures. JavaScript, also known as ECMAScript as an international standard, is a programming language used extensively in Web applications that has support for a data structure known as an "object" or data structure which can contain named properties or attributes ("ECMAScript Language Specification," 2011). In the JSON format each object is delimited by curly braces (Bray, 2014). Each attribute of an object consists of a name enclosed in quotes, a colon character, and the value (Bray, 2014). The value can be a number, a string of characters enclosed in quotes, an array, a Boolean (true/false) value, another object, or "null," which indicates a missing value (Bray, 2014). A JSON array is enclosed in brackets and contains a list of values (Bray, 2014). An object, an array, or a value can stand alone and is considered valid JSON (Bray, 2014).

According to Sassaman, Patterson, Bratus, and Shubina (2011), the JSON format is one of the most promising for encoding and transmitting structured data. Given the seriousness of the data validation issue and the increasing prevalence of JSON, this research will focus on the detection of data attacks on the JSON format.

Detecting attacks is typically performed by Intrusion Detection Systems (IDSs). These were first proposed in the 1980s (Denning, 1987). As the name implies, IDSs are intended to detect an intrusion by an attacker. Two common IDS types are known as "signature-based" and "anomaly-based" (Jiang, Song, Wang, Han, & Li, 2006). A related concept is that of the Web Application Firewall (WAF), which is also intended to detect or prevent attacks, but has much greater knowledge of the specifics of attacks on web applications (Trost, 2010). Signature-based detection systems must be programmed to recognize an attack based on a specific pattern of data or activity known as a "fingerprint" or "signature." Previously unknown attacks are less likely to be detected. Conversely, anomaly-based detection systems attempt to determine if something out of the ordinary is observed, rather than looking for a previously known attack signature. Given the threat of new attacks, detection by looking for anomalies or departures from expected data may be more effective. The focus of this research is the anomaly-based detection of malicious content in Web applications in which the JSON format is used to send commands or transfer data.

Finally, there are many examples of JSON being used outside of Web applications. The popular Elasticsearch search engine stores its data in JSON format (Gormley & Tong, 2015). The CouchDB database also uses JSON as its storage format (Anderson, Lehnardt, & Slater, 2010). The Ansible configuration tool, which is used for managing fleets of systems, uses a superset of JSON as its scripting language (Hochstein, 2014). Since JSON is used in many different ways, the results of this research could potentially be extended to a number of other applications of the JSON format.

Statement of the Problem

It can be difficult for a Web application to validate data sent to it from a client, particularly if such validation was not addressed in the design of the application. According to Ingham & Inoue (2007), anomaly detection for the HTTP protocol used by Web applications and services has been investigated. However, a search of existing scholarly publications has revealed little to no research that examines using multiple concurrent anomaly detection methods on both JSON structure and attribute values.

Nature and Significance of the Problem

Web applications provide functionality to authorized users, but they can also be leveraged by attackers who seek to cause potential harm. For instance, a Web application can accept commands in JSON format. An attacker may be able to send commands to the Web application that causes it to malfunction or to carry out some other harmful action. Furthermore, data coming back to the Web browser or client program may have malicious data inserted into it, which may cause harm to that client system. Due to the significance of this problem, this research will focus on the detection of attacks on JSON structured data.

Objective of the Research

The objective of this research is to evaluate a novel approach to the anomaly-based detection of attacks on Web applications and Web services that are launched via malicious modification or falsification of JSON structured data. This approach involves an ensemble of classifiers detection method utilizing data fusion from multiple measures that are potential indications of an attack. Classifiers are elements of software with the ability to categorize data categories (Han & Kamber, 2006). The form of data fusion used in this research is an approach in which measures of multiple attributes are taken across the same data for improved decision making ability. Boudjemaa and Forbes (as cited in Mitchell, 2007, p. 5) call this "fusion across attributes."

Hypothesis and/or Research Question(s)

Research Questions:

- Consider a mixture of the following attack types, which will be described in detail in Chapter 2, in which each type of attack occurs with approximately the same frequency or at least in sufficient numbers to represent real attack scenarios. There will be an explanation of the assumptions about the proportions of attacks later when Assumption 15 is introduced. Can an ensemble of JSON data structure and JSON attribute value anomaly detection classifiers implemented as a Random Forest classifier, which will be also described in Chapter 2, be used in a data fusion approach to classify data instances as attack and non-attack at a rate that exceeds chance expectation?
    - Buffer Overflow
    - JavaScript/HTML Insertion Attack
    - Command Injection
    - SQL Injection Attack
    - JSON Attribute Name Change
    - JSON Attribute Added
    - JSON Attribute Removed

    - JSON Attribute Value Manipulation

- Can attack detection occur at rates better than chance expectation as a function of different proportions of attack and non-attack instances during the time in which the classifiers are trained and in which the classifiers are tested? Furthermore, is this also the case for the number of classifiers within the Random Forest ensemble classifier?
- Specifically, will this be true for experimental combinations of training and test attack proportions of 5% to 95% in increments of 5%, as well as numbers of classifiers ranging from 1 to 95 classifiers in increments of 5 classifiers, with the exception of the first two values being 1 classifier and 5 classifiers?

H_0 (Null Hypothesis):

For a mixture of JSON data instances consisting of non-attack instances and instances affected by the attack types listed above, instances will not be correctly identified as non-attack or attack by a Random Forest classifier at a rate that is better than chance random expectation for any experimental permutations of attack instance prevalence during training, attack instance prevalence during testing, and number of classifiers.

H_1 (Alternate Hypothesis):

For a mixture of JSON data instances consisting of non-attack instances and instances affected by the attack types listed above, instances will be correctly identified as non-attack or attack by a Random Forest classifier at a rate that is better than chance random expectation for at least some experimental permutation of attack instance prevalence during training, attack instance prevalence during testing, and number of classifiers.

Limitations and Delimitations

There are limitations to this research:

- While a wide variety of examples of JSON data structures were used in this research, it was not possible to represent the entirety of data that could be represented in the JSON format.

Additionally, there are a number of delimitations:

- Only syntactically valid JSON data were in scope for this research. Instances of attack data are expected to contain structural anomalies, but all data had correct JSON syntax.
- Attacks on JSON that manipulate it with the intent that tampering will cause the modified data to be detected and rejected were not specifically addressed in this research.
- The chronological order or exact arrival time of JSON data structures in a stream of Web application data was not considered in scope for this research.
- Patterns of attacks involving multiple JSON data instances in sequence were not considered in this research. As an example, recognizing the pattern of a heap spray attack followed by an attack that triggers an error and activates the result of the heap spray is considered out of scope. Each attack is considered in isolation, even if it may be only one part of an attack with multiple parts.
- The data used by the researcher as well as code for representative attacks are not subject to any legal restrictions as used in this research, such as copyright infringement or prohibitions against reverse-engineering.

- Only JSON data directly in HTTP requests or responses that are not otherwise packaged, such as being included in another file that is being uploaded or downloaded, were in scope for this research. As an example, JSON data structures embedded in JavaScript files were considered out of scope for this research.
- All JSON data used in this research were in the UTF-8 Unicode format, which is the default for JSON (Bray, 2014).
- Simulated attacks on a JSON data instance were only made on one attribute or value at a time.
- In using the n-gram detection method, only character-level n-grams were utilized in which n is a maximum of five characters.

Definition of Terms

Anomaly: An anomaly is any value of a parameter that is unexpected (Jiang et al., 2006).

Attack: An attack is a credible attempt to compromise an asset (Gregory, 2010).

Byte: A byte consists of eight bits consisting of ones and zeros or on and off states (Hsu, 2012).

Character: A character can consist of one or more bytes (Hsu, 2012). In this research, a character implies a character in the Unicode encoding, which may be one or more bytes in length (Hsu, 2012).

Classification: Classification is assignment of data instances to categories (Han & Kamber, 2006).

Classifier: A classifier is a mechanism for performing classification.

Compromise: To compromise an asset is to harm its value (Landoll, 2011).

Confusion Matrix: A confusion matrix is a table which summarizes the possible outcomes of a classification process (Tan et al., 2006).

Cross Site Scripting (XSS): An XSS attack occurs when data entered by a user are allowed to contain JavaScript that can aid in stealing session information or other data (Howard, 2009; Watson, 2007).

Data Fusion: Data fusion is an approach in which multiple measures are used across the same data for improved decision making ability (Mitchell, 2007).

Decision Tree: A decision tree is a tree representation of a decision model (Han & Kamber, 2006).

Deterministic: Deterministic behavior is predictable (Zenil, 2011).

Encryption: Encryption is a process by which data is converted to a form that is not easily readable by an unauthorized person (Bangia, 2010).

Extensible Markup Language (XML): XML is a structured data format (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2008).

Hyper Text Markup Language (HTML): HTML is a document format for a system of linked Web pages and interactive menus (Berners-Lee & Connolly, 1993).

Instance: An instance is a member of a class or category which contains attributes common to the class or category (Han & Kamber, 2006).

Intrusion Detection System (IDS): An intrusion detection system detects an attack on a system (Denning, 1987).

JavaScript Object Notation (JSON): JSON is a text format description of a collection of JavaScript data structures (Bray, 2014).

Machine Code: Machine code consists of binary instructions that can be directly executed by a processor (Bangia, 2010).

MCC: The Matthews Correlation Coefficient (MCC) is a metric for evaluating the performance of a binary classifier (Baldi, Brunak, Chauvin, Andersen, & Nielsen, 2000).

Non-deterministic: Non-deterministic behavior is unpredictable (Zenil, 2011).

ROC: The Receiver Operating Characteristic (ROC) curve is a tool for evaluating classifier performance (Powers, 2011).

Shellcode: Shellcode is another term for machine code that is used to launch an attack.

String: A string consists of zero or more characters.

Structured Data: As defined in this research, structured data are expressed as letters, numbers, and punctuation and has a structure that is more complicated than a simple list of name and value pairs.

Supervisory Control and Data Acquisition System (SCADA): A SCADA system is technically one part of an industrial automation network, though it is commonly used to describe industrial control systems as well, which are also part of an industrial automated network (Knapp, 2011).

Test Set: A test set consists of pre-categorized data instances that are used to evaluate the performance of a classifier (Han & Kamber, 2006).

Training Set: A training set consists of pre-categorized data instances that are used to train a classifier (Han & Kamber, 2006).

Vulnerability: A vulnerability is a weakness that can be taken advantage of to compromise an asset (Gregory, 2010).

Web Application: Web applications are programs that respond when a user visits a Web page or when a program fetches information from a Web service.

Web Service: Web services are similar to Web applications, but are typically used for program to program communication.

Assumptions

Assumption 1:

JSON data structures sent as part of an attack differ in content, structure, or both from JSON data structures which are not part of an attack.

Assumption 2:

The more dissimilar a JSON data structure is from previously seen non-attack instances, the more likely that it is an attack.

Assumption 3:

The baseline training data contain examples of all valid structural permutations seen in JSON data structures sent to the server.

Assumption 4:

It is known which attribute names of JSON data structures can be safely mapped onto a single name, such as in the case that numeric ids occur in the dotted notation, which will be explained in Chapter 2, describing the attribute.

Assumption 5:

If elements of the JSON structure are mapped onto a single attribute name, it is acceptable to replace individual calculated measures, such as Shannon entropy, string length, numeric value, and n-gram counts of the ambiguous attributes with average values in a single attribute onto which the multiple attributes are mapped.

Assumption 6:

JavaScript insertion attacks, including XSS attacks and heap spraying attacks, may be adequately simulated by fragments of JavaScript and HTML code.

Assumption 7:

SQL injection attacks may be simulated by the inclusion of SQL attack code fragments in JSON object attributes.

Assumption 8:

Random number sources used in the experiment are uncorrelated.

Assumption 9:

Random votes in a majority decision voting process result in a random majority vote that is statistically equivalent to a single random voter.

Assumption 10:

If specific attacks and values are relatively unique to a given Web site, then this fact was taken into account by the Random Forest algorithm if data instances were tagged with a unique identifier for each web site.

Assumption 11:

Command injection attacks may be simulated by the inclusion of samples of system-specific attack commands from various operating systems.

Assumption 12:

Buffer overflow attacks may be simulated by including representative executable binary attack code encoded in the form of characters and placed in JSON string variables.

Assumption 13:

Multiple types of attacks may be simultaneously launched against a Web application or service. Therefore, all training and test data will contain multiple attack types rather than having separate data sets with different attack types in isolation.

Assumption 14:

A mixture of between 5% and 95% attack and non-attack instances is valid for both the training and test sets and will result in similar classification performance on datasets in which the attacks are real and not simulated.

Assumption 15:

Of the attacks used in the test and training sets, a composition of approximately equal portions of each attack type is valid and will result in similar classification performance on datasets in which the attacks are real and not simulated. If this is not possible due to a lack of data instances susceptible to specific attack types, then there will still be in aggregate a sufficient number of each attack type to represent real attacks. This is valid since the same dataset, if under real attack, would not be equally vulnerable to all attack types.

Assumption 16:

For the purposes of anomaly detection, instances of JSON data from a particular Web site are more relevant with respect to each other than JSON data from a different Web site.

Assumption 17:

Some measures of attack and non-attack instances may be identical. For example, for n-gram analysis, attack and non-attack instances could potentially have the same n-grams.

Assumption 18:

The Web browser producing the HTTP Archive (HAR) file used as the source for non-attack JSON data is assumed to be capable of recording HTTP requests with sufficient fidelity as to not be a source of experimental error.

Assumption 19:

Random number sources used in the experiment are sufficiently random as to not invalidate the experimental results.

Assumption 20:

Random classifiers based on these random number sources have discrete uniform distributions of true and false outputs.

Assumption 21:

There are no defects in the tools or software used in this research that would materially affect experimental outcomes.

Organization of Chapters 2 - 5

Chapter 2 will provide an overview of background information necessary for this research. Furthermore, a literature search of existing attack detection methods will be reviewed. Chapter 3 will describe the research methodology used. Chapter 4 will present and discuss the results of the research. Finally, Chapter 5 will conclude with a summary and discuss potential future research related to this work.

Chapter 2: Background and Literature Review

Introduction

This chapter will provide necessary background information. First, how JSON formatted data is typically transferred is explained in order to give a setting for the research. Structured data formats are described, with more detail given to JSON as well as its relation to the XML format. Web application attacks are explained. An overview is given of how other researchers have explored methods for attack detection in similar, though not identical, contexts as this research. The approach taken for anomaly detection is detailed, along with the pertinent metrics for attack detection. Furthermore, the Random Forest ensemble classifier is described.

HTTP Protocol

The exact transfer mechanism is of somewhat lesser importance for this research, since the content of the flow of structured data is what is being examined for anomalies, rather than a specific way the data are transferred. Nevertheless, in the interest of providing context, the Hyper Text Transport Protocol (HTTP) will be discussed as a method for transferring structured data. Also, though HTTP is referred to here, this is also intended to include the encrypted version of HTTP, which is HTTPS (Rescorla, 2000).

As Fielding et al. (1999) explain, the HTTP protocol, which is used by Web browsers to fetch and display pages from Web applications, has defined request and response phases. The requester, usually a Web browser, sends an HTTP request to a Web server and receives an HTTP response back. The HTTP request contains a “message header” section, followed by a “message body” section. When making an HTTP request, it is possible to pass parameters that will be used by the Web server in generating a response. These parameters may be found in either the message header or message body section. Unless JSON or some other structured data format is being used, data sent by a client to a Web server consists of data that is roughly in a list of name-value pairs (Raggett, Le Hors, & Jacobs, 1998). The HTTP request message header may contain additional information about the Web client or browser as well as items of data to maintain the Web application session state such that different HTTP requests are associated with the same user (Kristol & Montulli, 2000). The HTTP request message body is of particular interest to this research since it can contain structured data being sent to the server. The HTTP header may also contain structured data. For example, JSON formatted data could be included as a message header parameter. However, this will not be specifically addressed in this research since it is less commonly done and because the methods explored therein can be applied to JSON data, no matter how it is transferred. The HTTP response also contains a message header section, followed by a message body section, which contains such things as HTML pages, images, or executable code that will be utilized by the Web browser. This response can also contain structured data.

Structured Data: XML and JSON

Having set the stage for the exchange of structured data, a more comprehensive description of such data is in order. As previously mentioned, two common structured data formats are XML and JSON. Although this research deals with JSON, a discussion of both JSON and XML is worthwhile since XML can be converted to JSON and thus benefit from an ability to detect anomalies in the converted data. Both can be viewed conceptually as a “tree” of nodes. Figure 1 illustrates an example of a list of two street addresses represented as a tree. In the example in Figure 1, the AddressList node is the parent of the Address nodes. The Street, City, State, and Zip nodes are children of the Address nodes. The values of these nodes are various street, city, and zip code values.

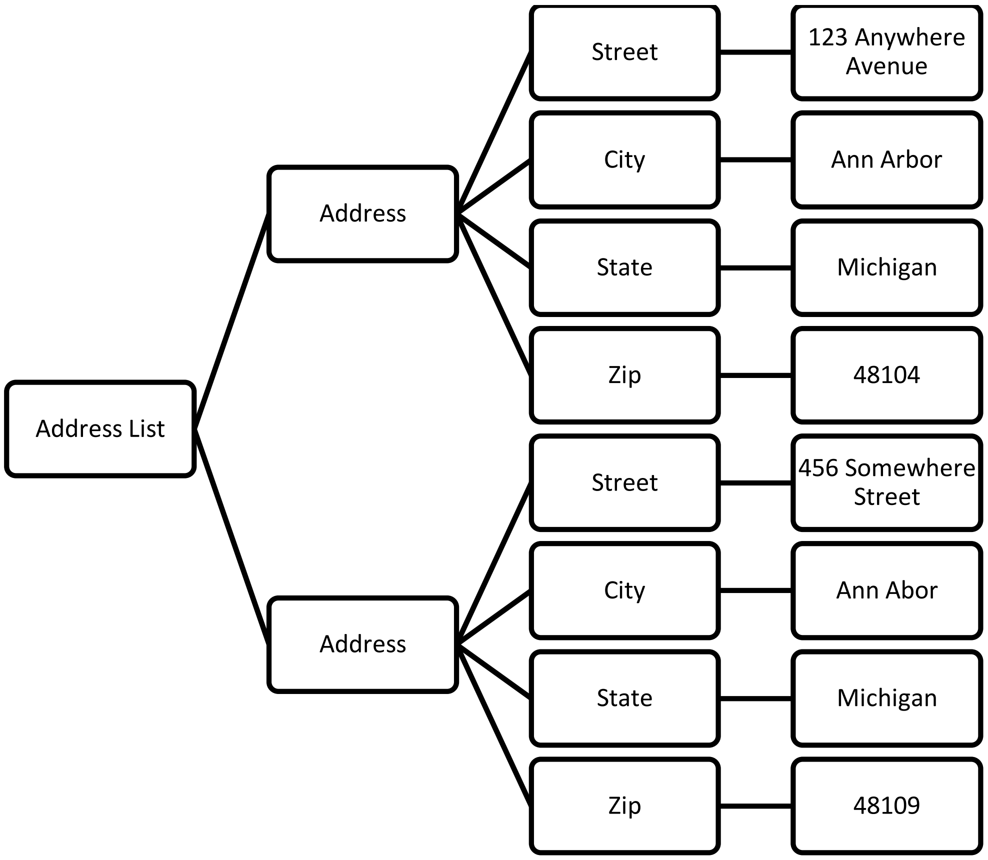


Figure 1. Address List Tree Example

The tree in Figure 1 can be represented in XML as a series of hierarchical nodes enclosed by the "<" and ">" characters. The start of a node is signaled by the node name, with the end of the node denoted by a leading slash in front of the node name. Here is the XML representation of the above tree:

```
<AddressList>
        <Address>
                <Street>123 Anywhere Avenue</Street>
                <City>Ann Arbor</City>
                <State>Michigan</State>
                <Zip>48104</Zip>
        </Address>
        <Address>
                <Street>456 Somewhere Avenue</Street>
                <City>Ann Arbor</City>
                <State>Michigan</State>
                <Zip>48109</Zip>
        </Address>
</AddressList>
```

The tree in Figure 1 can also be represented in JSON:

```
{ "AddressList": {
        "Address": [
                { "Street": "123 Anywhere Avenue",
                "City": "Ann Arbor",
                "State": "Michigan",
                "Zip": "48104" },
                { "Street": "456 Somewhere Avenue",
                "City": "Ann Arbor",
                "State": "Michigan",
                "Zip": "48109" }
        ]
}}
```

XML has an advantage in that a given document may be validated against a schema, or set of rules, governing its structure (Lee & Chu, 2000). A similar mechanism for JSON was proposed in an Internet Engineering Task Force (IETF) draft that has since expired (Galiegue, Zyp, & Court, 2013). Schema validation can prevent an attacker from trying to deliver malicious content that is structurally or semantically different than valid data. Unfortunately, its use is not widespread. Furthermore, Lampesberger (2013) reports that as of 2013 only 8.9 percent of XML documents in the Web validate to a schema. If JSON schema validation adoption follows a similar path, it may be quite some time before widespread adoption occurs. Moreover, legacy software may not adopt schema validation at all. Therefore, being able to detect attacks on the structural integrity of JSON data, independent of Web application implementations is worthwhile since it may be done without expending resources to modify the Web application itself.

An advantage of JSON over XML is that it is widely supported without requiring the use of add-on software libraries (Severance, 2012). Also, as in the above example, XML structures can be more verbose than their equivalent JSON counterparts. This verbosity is inconvenient for some applications. Additionally, XML can generally be converted to equivalent JSON. From the above example, a single address node could have been alternatively expressed as:

```
<Address Street=”123 Anywhere Avenue” City=” Ann Arbor” State=”Michigan”
                    Zip=”48104”/>
```

However, this can be seen as simply moving the values that were formerly contained in JSON child nodes instead to be within the “<” and “>” of the XML parent node. Because XML data can be converted to JSON for analysis, this research will focus on JSON.

Web Application Attacks

It is worth considering the sort of attacks that can be made on Web servers via tampering with JSON formatted data. Since JSON formatted data are sent to a Web server via HTTP, some of the same attacks that have formerly been used against other formats of commands and data sent via HTTP could be used against JSON as well.

Halfond, Viegas, and Orso (2006) describe security issues that arise when HTTP query parameters are not screened for malicious values that may be directly included in Structured Query Language (SQL) statements. The text included in the attack is fragments of SQL language statements. Attacks based on this principle are known as SQL injection attacks. If values from JSON are similarly utilized by a Web application without validation, the same type of attack could be launched as effectively as SQL injection attacks.

Another attack is that of the buffer overflow (Watson, 2007). In the buffer overflow attack, a stream of data is created that will be copied into a destination that is too small to contain it. By carefully crafting this stream of data, it is possible to cause instructions to be executed by the attacked system. A related attack is that of the "heap spray" (Hsu et al., 2010). In this attack, a system is tricked into executing malicious code that fills memory with specially formatted data. The attack is successful when an error causes the attacked system to execute the data that was sprayed. In both of these attacks, the malicious data sent to the server can be in the form of text that is inadvertently converted by the Web server to executable instructions.

Another form of attack known as "command injection" can cause native commands to be executed on a victim system. If the victim system tries to execute strings passed to it via the data submitted to a Web application without any screening, it is possible to execute arbitrary commands on that system (Howard, 2009; Watson, 2007).

Yet another common attack is Cross Site Scripting (XSS) which occurs when data entered by a user is allowed to contain JavaScript that can aid in stealing session information or other data as well as can be used to otherwise compromise the security of a Web application (Howard, 2009; Watson, 2007).

Invalid data or data that causes errors on the server can be another form of attack (Watson, 2007). Improperly handled errors can leave the server in an inconsistent, unsecure state. As an example, a value in a JSON object might be used as an array index without validation. The index could be changed to a value that caused a memory error, an array index out of bounds, or other unauthorized access or error.

The next question to answer is how a JSON object might be modified or crafted by an attacker to carry out the foregoing attacks. There are at least three basic ways this may be done:

- A syntactically invalid JSON object may be submitted to the Web server.
- The structure of a JSON object could be modified such that it is syntactically valid, but still causes an error. An unexpected attribute could be added, an expected attribute could be removed, the type of an attribute could change, or the number or types of elements in an array could vary. On the other hand, changing the order of the attributes in a JSON object cannot technically be considered an attack since the order of the attributes of a JavaScript object is undefined (Crockford, 2008).
- An attribute of the JSON object may be modified as part of an attack. This could include strings of characters, arrays of values, Boolean (true/false) values, or

numbers. Objects can be attributes as well, but changing an entire sub-object would be equivalent to modifying the structure of the JSON object structure. This is noteworthy since it means that different types of attacks can in some circumstances resemble each other.

The first method of attack, that of sending syntactically invalid JSON data, is relatively easy enough to defend against since it is possible to determine if JSON data are syntactically valid. There is a standard function in recent JavaScript implementations to safely consume syntactically valid JSON data without actually executing malicious code that may be embedded in it (Crockford, 2008). On the other hand, this is not to say that a failure to parse the data will be handled gracefully by Web application software. Since the defense against this attack can be simply to discard information that does not constitute valid JSON, it will not be considered in this research. It is possible for an attacker to use this method to cause important data to be rejected, but this particular attack is considered out of scope for this research.

The detection of the second type of attack, that of sending structurally modified, but syntactically correct, JSON, amounts to a comparison of tree structures. That is, a suspect tree structure may be compared to structures that are normally encountered in the use of the Web application. There are methods of tree comparison as well as tree structure mining (Han & Kamber, 2006). Furthermore, some researchers have derived formal language descriptions based on received XML data (Lampesberger, 2013). In this research, a simplified heuristic approach for labeling attributes is used for tree comparison. This labeling has features similar to JSONPath expressions (Windley, 2012). Using the previous example of an address list, a text description of the data structure may be formed by using a period character to concatenate nested attribute names, brackets to denote the elements of arrays, and the JavaScript type of the attribute in curly braces. This convention will be referred to here as "dotted notation." For the sake of simplicity, this ignores cases in which the attribute names themselves contain periods, brackets, or curly braces. Standard techniques for dealing with delimiters in attribute names, such as replacing them with alternate characters, may be used to avoid this problem and will not be discussed here. Using this technique, the structure of the first address in the tree can be described as:

```
AddressList[0]{object}
AddressList[0].Address.Street{string}
AddressList[0].Address.City{string}
AddressList[0].Address.State{string}
AddressList[0].Address.Zip{string}
AddressList[1]{object}
AddressList[1].Address.Street{string}
AddressList[1].Address.City{string}
AddressList[1].Address.State{string}
AddressList[1].Address.Zip{string}
```

Essentially what has been done is that the position of an attribute within an object that has been encoded in the JSON notation, along with its type, has been expressed as a single string of characters. This representation will remain the same, even if the values of strings, Boolean values, and numbers change. As an example, assume that over time, the above list was found to be common in all observed address lists. Further assume that a maliciously modified object was received which had the following structure:

```
AddressList[0]{object}
AddressList[0].Address.Street{string}
AddressList[0].Address.City{string}
AddressList[0].Address.State{string}
AddressList[0].Address.Zip{string}
AddressList[0].Address.Attack{string}
AddressList[1]{object}
AddressList[1].Address.Street{string}
AddressList[1].Address.City{number}
AddressList[1].Address.State{string}
AddressList[2].Address.Street{string}
AddressList[2].Address.City{string}
AddressList[2].Address.State{string}
AddressList[2].Address.Zip{string}
```

Upon examination, a number of issues become apparent. First, there is a missing AddressList[1].Address.Zip element in the second address. Second, there is an unexpected element AddressList[0].Address.Attack element in the first address. Third, an entirely unexpected third address AddressList[2] that has been added. Fourth, the AddressList[1].Address.City{number} element can be seen to have an unexpected type. It was formerly a string of characters, but now a numeric value is present. Detection of these differences can be seen as a simple comparison of the elements of two sets. The measure of similarity between these two sets may be calculated as the Jaccard coefficient, which is a metric ranging from 0 to 1 as to how similar the sets are to each other (Han & Kamber, 2006). The foregoing assumes (Assumption 1) that attacks will differ in a detectable way from normal data. Consideration must also be given to the possibility that attributes of a JSON structure change dynamically over time and should not be classified as an attack. In this case it would be a false positive to identify this as an attack. For this research, the assumption (Assumption 2) is made that the more dissimilar a JSON data structure is to non-attack examples that have been seen previously, the more likely it is to be an attack. The Jaccard coefficient of a newly observed JSON data structure calculated with respect to those previously seen is used for the detection of structural anomalies.

Had the above example been more complicated in that AddressList had contained mixed types that occurred in different orders, or that one of the Address attributes was typically missing, but was permitted in either the first ([0]) or second ([1]) position of the array, then the comparison would have been less straightforward. The root difficulty in these cases is related to consistent order in the array. On the other hand, if baseline examples contained all permutations of where elements in the array might occur, then there would be a match. However, this might be difficult in that data structures with multiple arrays could have a large number of permutations seen infrequently in actual practice. One possible approach would be to sort the elements in JSON arrays by their value before generating the dotted notation description. Unfortunately, this could mask an attack that was made by exchanging the elements of an array in an attempt to cause a malfunction. Therefore, it will not be adopted. Consequently, for this research, it is assumed (Assumption 3) that baseline examples will contain all valid permutations.

There is another complicating issue with regard to detecting structural tampering. In a sampling of data taken as this research was being planned, it was discovered that attribute names, rather than just the attribute value, could contain unique identifying values. For example, in the dotted notation used in this research, some of the elements used in JSON on one of the Wikimedia Foundation Web sites as found among several JSON object were as follows:

- query.pages.3742775.pageid{number}
- query.pages.43240234.pageid{number}
- query.pages.36596570.pageid{number}
- query.pages.766068.pageid{number}
- query.pages.35057023.pageid{number}

The integer value will guarantee that each attribute is treated as a unique attribute, which would prevent comparison between them that might otherwise occur. This may or may not be appropriate, depending on the Web application. Other datasets not used in this research may have different conventions for unique identifiers, such as mixed letters and numbers. While it is likely possible to automate an approach to locating unique identifiers that occur in JSON attribute names, an assumption (Assumption 4) made for this research is that the attribute names in the structure that may be safely treated as a generic match are known in advance. For example, all of the above dotted notations would be collapsed into a single attribute name "query.pages.*.pageid{number}." This will cause all of these attributes to essentially be treated as the same attribute. For the data used in this research, this is simply performed by translating any component of an attribute name that is an integer to an asterisk "*" or wildcard match. If collapsing these attributes into a single attribute causes ambiguity within a JSON instance, it is assumed (Assumption 5) that calculated values, such as Shannon entropy, string length, numeric value, or number of n-grams, can be replaced by the average values of the ambiguous attributes. As will be subsequently described, these calculated values are used to determine whether or not an instance of JSON data is an attack.

The third type of attack, that of modifying a JSON attribute value, is perhaps the most difficult to detect. If an attack causes a change in the type of the value, such as a SQL injection attack presenting as a string value in place of what was formerly a numeric value, then this will be trivial to detect as a structural anomaly. But assuming that a type change does not result, detection will be more difficult. The detection of anomalies in each of the possible data types will be considered here in order of difficulty.

Detecting an unusual numeric value could potentially be accomplished via statistical measures or range checking. However, there is an argument for using non-parametric data mining techniques since these make no assumption about the data following any specific distribution, such as being normally distributed, as opposed to statistical anomaly detection techniques which do (Tan, Steinbach, & Kumar, 2006). Moreover, statistical outlier tests for multivariate data can perform poorly (Tan et al., 2006).

Continuing with the third attack type, strings of characters pose special problems. These problems may be illustrated with a simple natural language example. Consider the phrase “the quick brown fox.” Is this an attack? Perhaps the acceptable words for this string are in Italian. In this case, the phrase is definitely unexpected. On the other hand, perhaps all foxes must be red. In this case, “brown” is an unexpected fox color. An attack may not be as blatant as the one-line JavaScript program “alert(‘This is an XSS attack!’)” being sent, which would trivially indicate that an attacker was trying to exploit a cross site scripting (XSS) vulnerability. Consequently, it is difficult to determine the full width and breadth of possible data values since a given Web application can have restrictions that may be difficult to know without extensive knowledge of its design or the behavior of its users. In a Web application with limited functionality, it may be possible to predict all values for the data that may be found in JSON objects sent to and from a Web server. This data would be said to be deterministic or predictable (Zenil, 2011). However, a more complex Web application could also have data that is non-deterministic or unpredictable (Zenil, 2011). Since predictability of the data is not a certainty, a method must be adopted which addresses either case. For attributes that are strings, the working assumption (Assumption 1) is that XSS, SQL injection, buffer overflow, or heap spray attacks will differ substantially in content from strings that are not part of an attack. Kruegel, Vigna, & Robertson (2005) compared attack strings in HTTP submissions to idealized character distribution (ICD) for the parameters gathered under normal (non-attack) conditions. That is, the frequency of characters in a string to be examined for attack was compared to statistical distributions learned under training conditions. Kreugel et al. (2005) as well as Choi, Kim, Choi, and Lee (2009) found HTTP parameter length to be significant in detecting attacks. Although HTTP parameter length is not identical to the length of a string value in a JSON object, this research will explore JSON string value length as a measure. Actual HTTP parameter length is ignored in this research since the length of JSON data contained within HTTP parameters would be directly related to HTTP parameter length. Character or byte-level n-gram frequency has been used before for anomaly-based attack detection (Wang, Parekh, & Stolfo, 2006; Wressnegger, Schwenk, Arp, & Rieck, 2013; Choi et al., 2009; Choi, Choi, Ko, & Kim, 2012). An n-gram is simply n characters or words that occur together. Continuing with the previous example of “the quick brown fox,” some example character n-grams would be “t,” “th,” and “the” for n being 1, 2, and 3, respectively. Example word level n-grams would be "the," "the quick," and "the quick brown fox," again for n being 1, 2, and 4, respectively. Word level n-grams will not be used in this research, but are mentioned here for completeness. This research will use character n-gram analysis for the anomaly-based detection of attacks on string values in JSON data structures.

In an approach adapted from Lyda & Hamrock (2007) and Choi et al. (2009) Shannon entropy will also be used in this research to differentiate between attack and non-attack data for text values of attributes. Shannon entropy is a measure of the amount of information that is present in a group of symbols. From Shannon (2001, p. 19) the information entropy, known as H, can be estimated as the following where p_i is the probability of a given symbol, n is the number of unique symbols, and the occurrence of each symbol is assumed to be statistically independent:

$$H = -\sum_{i=1}^{n} p_i * \log_2 p_i \tag{1}$$

This value is an estimate since the order of different symbols may be statistically significant and require evaluation of the probabilities of groups of symbols occurring together (Shannon, 1951).

At this point it is worthwhile to consider the nature of text that may be present in an attack on a JSON string attribute. XSS attacks will consist of a mixture of JavaScript and potentially HTML. SQL injection attacks will contain what could be part of a valid SQL string. Buffer overflows will consist of overly long strings than can be translated into processor instructions. And finally, heap spray attacks that may occur as strings within JSON objects will consist of JavaScript code perhaps mixed with HTML. Therefore, the assumption (Assumption 6) is made that JavaScript insertion attacks, including XSS attacks and heap spraying attacks, may be adequately simulated by fragments of JavaScript and

HTML code. Such attacks will be referred to here as JavaScript Insertion attacks. Obfuscated JavaScript, or JavaScript manipulated in ways to avoid detection, will also be included in the JavaScript Insertion attacks (Choi et al., 2009). Furthermore, it will be assumed (Assumption 7) that SQL injection attacks may be simulated by the inclusion of SQL attack code fragments in JSON object attributes.

The remaining data type is that of the Boolean. This is somewhat challenging. If a Boolean variable is present, in order to be meaningful, it must take on either false or true values at some point. A Boolean variable that is always true or always false communicates no information (Shannon, 2001). On the other hand, it cannot be assumed that a constantly true or false Boolean is abnormal. The particular value the variable takes on must be taken in context with the other variables in a given JSON object.

Several different types of attacks have been mentioned, along with methods that could be used to detect anomalies indicating these attacks. Since there are multiple attack types and ways of detecting them, it is reasonable to speculate that one single approach may not be the best to detect all attacks. Rather than relying on a single measure, a potentially superior approach is described by Boudjemaa and Forbes (as cited in Mitchell, 2007, p. 5) as "data fusion across attributes" that combines different measurements across the same data. Since this can support a more robust data analysis, it is the intent of this research to use multiple measures, such as structural anomalies as well as JSON attribute value anomalies. The approach used in this research was to analyze the JSON data being transmitted in a Web application, using the simultaneous application of several different attack detection mechanisms in an effort to outperform any single attack detection approach.

Detection Method - Classification

The next topic to address is how to combine the different potential indications of attack into a single indicator for a given JSON data structure. Classification, or assignment to categories, such as “attack” and “non-attack,” is predicated on the existence of categories as well as some process for assigning instances, which can be thought of as rows in a table of data, to those categories (Han & Kamber, 2006). In the case of this research, each instance corresponds to an observed JSON data structure.

Once categories or classes are established, the process for assigning instances to classes must be developed. This is performed through a process of “supervised learning” (Han & Kamber, 2006). A set of pre-categorized data instances called the training set is used to train the classifier, whatever classification algorithm is used, to output a particular category name given an instance of data from the training set. A test set of data instances, also pre-categorized, is then used to evaluate the effectiveness of the classifier (Han & Kamber, 2006). The classifier may not be 100% accurate. The degree of accuracy required depends on the particular desired use of the classifier and may involve tradeoffs in such things as speed of classification (Masud, Khan, & Thuraisingham, 2012). Confusion matrices, which are a compact means of displaying predicted versus actual values, may be used to summarize the accuracy of the classifier (Han & Kamber, 2006). A confusion matrix may be seen in Table 1. The column and row headers indicate if an attack was actually present and if an attack was detected, respectively. The intersections of columns and rows contain a count of all four possible permutations of attack presence and detection.

Table 1
Confusion Matrix

	Attack Present	**No Attack Present**
Attack Detected	Number of True Positives	Number of False Positives
No Attack Detected	Number of False Negatives	Number of True Negatives

A true positive is a correctly identified attack. A true negative is a correctly identified non-attack. A false negative means that an attack was present, but it was not identified. A false positive means that an attack was identified, but none was actually present. False negatives can be of concern since an attack is missed. However, if false positives occur in large enough numbers, they can wastefully consume significant resources since they must be investigated and not yield any benefit (Collins, 2014). Excessive false positives may also point to overall problems with the classification of attacks.

An important detail is a choice of classification algorithm. The Random Forest algorithm chosen for this research was introduced by Breiman (2001). According to Hastie, Tibshirani, and Friedman (2009), the Random Forest algorithm requires little configuration, is resistant to noisy data, and tends not to overfit. These are typical challenges that other classification algorithms face. In particular, the resistance to overfitting is important since overfitting can cause a classifier to learn small variations in the training data and thus lose classification ability when presented with data on which it has not trained (Bhattacharyya & Kalita, 2014). Considering that the approach in this research is for anomaly, rather than signature detection, having a classifier being able to classify previously unknown data is essential.

Random Forests

The Random Forest algorithm uses a large ensemble of randomly built decision trees, rather than a single decision tree. Figure 2 shows an extremely simple example of a single decision tree used to classify fruit. In a decision tree, regardless of whether the Random Forest algorithm is used, the decision may be based on qualitative attributes, as in the example, or ranges of quantitative variables.

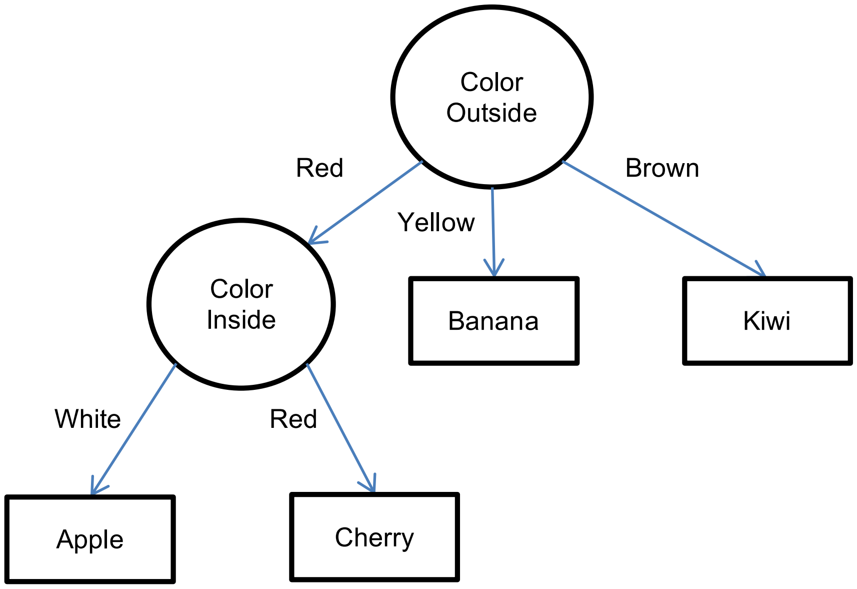


Figure 2. Decision Tree Example

In the Random Forest algorithm, each decision tree has a random selection of attributes and is trained on a random subset of the training data (Breiman, 2001). To expand upon the example in Figure 2, assume that in addition to “Inside Color” and “Outside Color,” “Shape” is also an attribute. Given this, another possible tree in the ensemble can be seen in Figure 3.

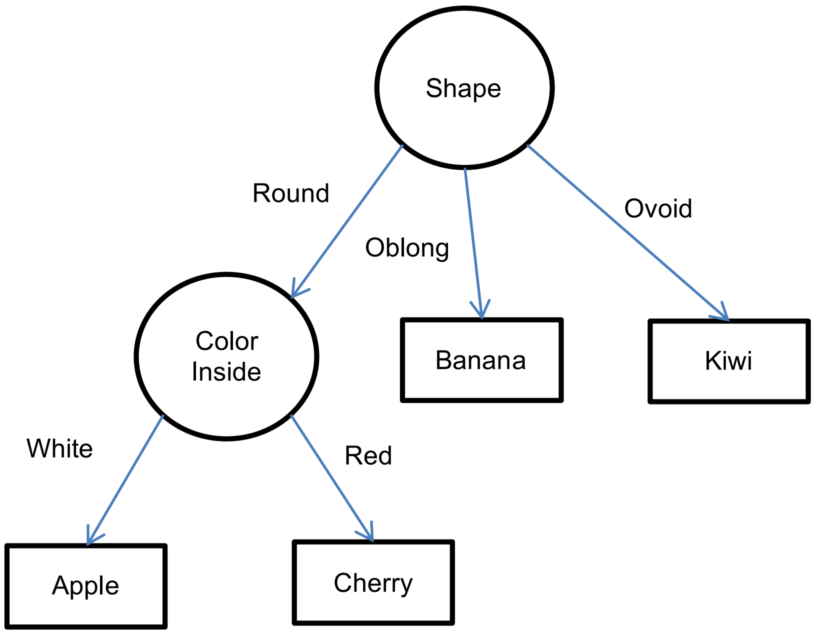


Figure 3. Alternate Decision Tree One

The construction of each of the trees, once a selection of attributes has been made, involves selecting the attributes in order as one moves down from the root of the tree which will result in the best classification. In Figure 3, if the “Color Inside” attribute had occurred at the root, a different tree would have resulted as in Figure 4.

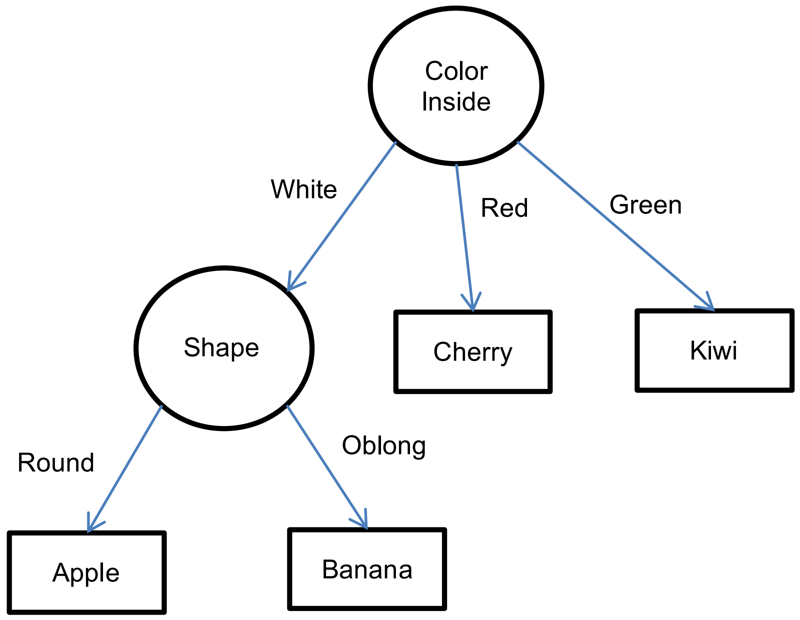


Figure 4. Alternate Decision Tree Two

If there were typically even numbers of bananas, kiwis, apples, and cherries, the trees in Figure 3 and Figure 4 probably offer no advantage over each other. However, if most of the fruit in the training set consisted of only bananas and kiwis, then the tree in Figure 3 would have classified the bulk of the fruit at the root node. On the other hand, Figure 4 would need to have examined the "Color Inside" attribute as well as the "Shape" before classifying the bananas. Given a predominance of bananas and kiwis, a classifier which asks the "Shape" question first will perform better. Two metrics commonly used in the Random Forest algorithm for determining the best attribute to use to split the data into distinct sets at each node are entropy, as defined earlier, while another is the Gini index (Hastie et al., 2009). Breiman (2001) specifies the use of the Classification And Regression Tree (CART) decision tree implementation, which by Robnik-Šikonja (2004) uses the Gini index. It is also the default metric in the Random Forest implantation in the Scikit-Learn software that was used in the implementation of this research (Pedregosa et al., 2011).

Summary

In this chapter background information was provided concerning the JSON format, how JSON is transferred via the HTTP protocol, possible attacks on JSON, metrics that can potentially indicate attack, and how these metrics can be used with a classifier to categorize instances of JSON format data as to whether or not they have been attacked. In addition, existing literature on attack detection methodology was reviewed. A description was given for how techniques were adapted from prior research in different settings, combined with new approaches, and implemented in a data fusion methodology using a Random Forest ensemble classifier.

Chapter 3: Methodology

Type of Research

The research methodology used is experimental, with a pre-test, post-test control group design (Leedy & Ormrod, 2010). Initially the experimental group and control group are on an even footing. Neither should be able to detect attacks at a rate better than chance expectation since neither is trained. The pre-test serves to confirm this. The treatment applied is the "training" that the experimental group receives to learn to detect attacks. The post-test evaluates whether or not the ability of the trained experimental group to detect attacks exceeds that of the control group.

Research Design

The organization of the experiment can be seen in Table 2. MCC is the Matthews Correlation Coefficient, which will be subsequently described. The experimental and control groups consist of classifiers defined as follows:

- If untrained, a classifier will randomly identify a JSON data instance as "attack" or "non-attack." Two arbitrary untrained classifiers will not necessarily have the same output for the same inputs as their outputs are assumed (Assumption 8) to be uncorrelated random variables. The experimental group is a random classifier prior to training. The control group, since it receives no treatment, remains a random classifier throughout the experiment. The assumption (Assumption 9) is that a single random classifier has an equivalent output to an aggregation of random classifiers participating in a majority vote. Therefore, the control group will consist of a single random classifier.

- If trained, a Random Forest classifier is used to identify instances as "attack" or "non-attack." The experimental group is a random classifier before training, but not after training. A single Random Forest classifier actually consists of an ensemble of randomly built classifiers which are used in a majority vote decision process. The intent of the Random Forest algorithm is to have a group of different classifiers that on aggregate perform better than any single classifier. Implementations of this algorithm typically allow any number of classifiers to be specified. In this way, the experimental group of the experiment can be built up simply by specifying the desired number of randomly built classifiers when constructing a single Random Forest classifier. In effect, randomization is built into the algorithm.

Table 2
Experimental Design – Pretest-Posttest

Group	Pretest	Treatment	Posttest
Experimental Group	MCC Test of Classification Using Test Dataset	Training with Training Dataset	MCC Test of Classification Using Test Dataset
Control Group	MCC Test of Classification Using Test Dataset	None	MCC Test of Classification Using Test Dataset

Depending on the trial, there were between 1 and 95 classifiers, in increments of 5 classifiers, though trials one and two will have 1 and 5 classifiers, respectively, within the instance of the Random Forest classifier in the experimental group. Both the training and attack percentage composition were allowed to independently vary between 5% to 95% attack instances, in intervals of 5%. The experiment was repeated for all permutations of attack proportions and number of classifiers for a total of 7,220 trials as given by Equation 2.

$$19\ Training\ Attack\ Percentages * 19\ Test\ Attack\ Percentages * 20\ Classifiers = 7{,}220\ trials \tag{2}$$

An assumption (Assumption 10) is that if specific attacks and values are relatively unique to a given Web site, then this fact is taken into account by the Random Forest algorithm by the inclusion of a unique web site id as one of the attributes used for constructing decision trees. This assumption allows data from all Web sites to be simultaneously evaluated.

Note that in the pretest, due to the preceding definition of the classifier, what is effectively being performed is a comparison of random number generators. If any statistically significant predictive ability or difference between the experimental group and the control group is found in the pretest, this will indicate problems with the source of random numbers used to arbitrarily classify instances in the untrained classifier. This will serve as a check on validity.

After the pretest, training for the experimental group occurred using a mixed set of JSON data instances. As specified previously, 5% to 95% of the instances were attack instances, depending on the iteration of the experiment. The attack instances consisted of approximately equal portions of each attack type, with exceptions addressed in Assumption 15.

If the classifiers in the experimental group correctly identify the attacks at a statistically significant greater rate than the random classifiers in the control group, then the experimental results fail to support H_0, the null hypothesis. There were two confusion matrices for each iteration of the experiment, one for the pre-test and one for the post-test. In addition to examining the performance for each of the percentages of attack instances, it is also desirable to have an overall result reflecting all trials of the experiment. The approach taken to measure overall result for all trials was to sum the entries in the confusion matrix in

Table 3 across all runs of the experiment separately for the pre-test and post-test confusion matrices. This effectively serves to average the performance across all runs with different compositions of attack types. Failure to support H_0, the null hypothesis, in this setting would serve to increase external validity since the results could be applied more broadly to different attack mixtures that may be found in actual practice.

Table 3
Confusion Matrix with Row and Column Totals

	Attack Detected		
Attack Present	**True**	**False**	
True	Number of True Positives (TP)	Number of False Negatives (FN)	Total Number of Actual Attacks
False	Number of False Positives (FP)	Number of True Negatives (TN)	Total Number of Non-Attacks
	Total Detected Positives	Total Detected Negatives	Total Observations (N)

The Matthews Correlation Coefficient (MCC) was used to determine if the classifiers in aggregate were detecting attacks at a statistically significant rate (Baldi, Brunak, Chauvin, Andersen, & Nielsen, 2000). This coefficient is used to measure the quality of binary classification, which in this case is whether or not an attack is present. Furthermore, the MCC has an easily applied statistical approximation using the chi-square statistic to determine if the predictive ability of a classifier is statistically significant Baldi et al. (2000).

From Baldi et al. (2000, p. 415), the basic formula of the MCC is:

$$MCC = \frac{TP\ x\ TN - FP\ x\ FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3}$$

Where

- TP = True Positive (actual attack, attack detected)
- TN = True Negative (no actual attack, no attack detected)

- FP = False Positive (no actual attack, attack detected)
- FN = False Negative (actual attack, no attack detected)

The MCC ranges between -1 and 1. A -1 means perfectly wrong classification, a 0 means no ability to classify or close to random accuracy, and a 1 means perfectly correct classification. This statistic has the added benefit that by Baldi et al. (2000, p. 415) it can be related to the chi-square statistic of the confusion matrix given in Table 3 by $\chi^2 = N * MCC^2$ where N is the number observations. Since the MCC^2 term will always be positive, as N, the number of observations, increases, the χ^2 value will also increase, thus the p-value will go down since there is a decreasing probability that the results were produced purely by chance. Likewise, if N is held constant and MCC increases, the χ^2 value will increase and the p-value will decrease. Therefore, as classification accuracy increases, the p-value decreases. The relationship of χ^2 and p-values for one degree of freedom is illustrated in Figure 5. As can be seen in the confusion matrix in Table 3, there is one degree of freedom since $degrees\ of\ freedom = (rows - 1)\times(columns - 1) = (2 - 1)\times(2 - 1) = 1$ (Blaikie, 2003).

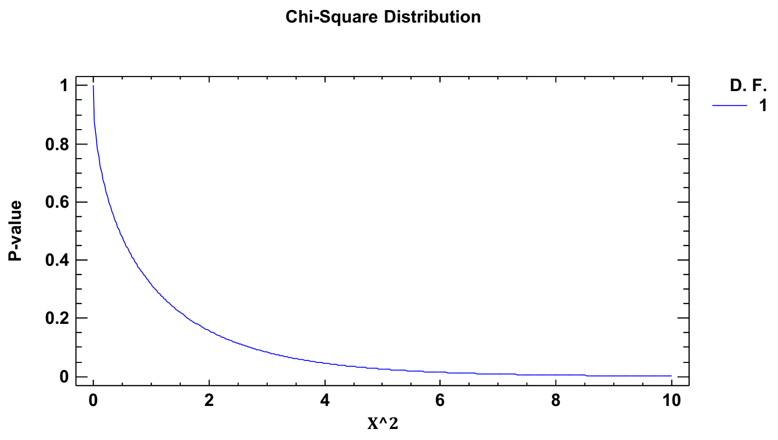


Figure 5. Chi-Square Distribution

The χ^2 statistic may be evaluated to determine if, at a given level of significance, the results were obtained purely by chance (Baldi et al., 2003). If some attacks are detected at a statistically significant rate, H_0 will be rejected. If the resulting p-value is $\leq .05$, the null hypothesis H_0 will be rejected in favor of the alternate hypothesis H_1. If the resulting p-value is > .05 the null hypothesis H_0 will fail to be rejected.

An alternative approach to the use of the MCC would be to use Receiver Operating Characteristic (ROC) analysis (Powers, 2011). However, by Powers (2011), ROC analysis does not take into account the number of predicted positives that is, both true positives and false positives, versus the number of actual positives. This said, ROC analysis was useful in this research for relative comparisons between classifiers.

Population and Sample

The population for this research consists of Random Forest classifiers. Some of the classifiers are untrained (the control group) and others (the experimental group) are trained on both non-attack and simulated attack JSON data instances. In this experiment, applying the training JSON data instances is a "treatment" since it will affect the characteristics of the Random Forest classifiers by exposure during the training phase.

A challenge with this research is to obtain JSON data instances that could be encountered in actual practice. Ideally this would be done by capturing actual JSON data from interactions with Web applications. Unfortunately, there are potential legal issues that make this difficult. Capturing and analyzing the structure of the JSON data could be considered reverse-engineering, which may not be allowed by the terms of use for a Web site. Furthermore, the JSON data could include copyrighted content that could not easily be made available to other researchers wishing to reproduce the results of this study. To address these concerns, JSON data was drawn from interactions with twelve Web sites operated under the auspices of the Wikimedia Foundation, Inc.:

- commons.wikimedia.org
- en.wikibooks.org
- en.wikinews.org
- en.wikipedia.org
- en.wikiquote.org
- en.wikisource.org
- en.wikiversity.org
- en.wikivoyage.org
- en.wiktionary.org
- species.wikimedia.org
- wikidata.org
- wikisource.org

The Wikimedia Foundation specifies that contributed contented be freely licensed in their terms of use ("Terms of Use - Wikimedia Foundation," n.d.). Furthermore, the software that is used in the operation of these sites is licensed under the GNU General Public License (GPL), which allows free access to and use of the software ("Manual:What is MediaWiki? - MediaWiki," n.d.).

It is important to note that all the simulated attacks in this research were conducted upon data which was gathered from interactions with the Web sites. The attack simulation was done offline, and in no way affected the operation of the Web sites or violated any terms of use for the sites which were the source of the data.

At least 200 JSON data instances were gathered from interactions with each site. As seen in Table 4, the Dataset ID field is used to indicate the particular Web site corresponding to each data instance, while the Instance ID is unique within a dataset. Although displayed here in a column, these elements actually form rows, one per each instance or JSON object in the dataset. The elements labeled as “All Instances” occur once per instance. The group of elements labeled “Value 1” through “Value N” will occur once for each unique JSON attribute that occurs in either the training or test datasets. That is, they are the union of all attributes found in the training and test datasets.

Table 4
Training and Test Data

All Instances	**Dataset ID**	Unique number between datasets
	Instance ID	Unique number within a dataset
	Web Site ID	Unique number between Web sites
	Attack Present	True/False (1 or 0) Note: This variable is used as a target category for training or as a verification variable during testing. Otherwise it would be trivial to detect an attack by examining this variable.
	Jaccard Coefficient	Numeric Value - Within dataset minimum Jaccard Coefficient distance from all non-attack instances in the training set based on presence or absence of attributes
	Overall Shannon Entropy	Shannon entropy calculated for the entire JSON instance
	Overall Non-Attack N-Grams Present	Non-attack n-grams present for the entire JSON instance and not just one attribute
	Overall Attack N-Grams Present	Attack n-grams present for the entire JSON instance and not just one attribute
	JSON Length	Length in characters of JSON instance
Value 1	**Value**	If Boolean True=1, False=0 If Numeric, Numeric Value If String =length in characters If Array/Object=0
	Number of Attack N-Grams Present	Calculated numeric value for strings If type of attribute is not string, value=0
	Number of Non-Attack N-Grams Present	Calculated numeric value for strings If type of attribute is not string, value=0
	Shannon Entropy of Attribute Value	Shannon entropy for strings If type of attribute is not string, value=0
	…	**…**
Value N	**Value**	If Boolean True=1, False=0 If Numeric, Numeric Value If String =length in characters If Array/Object=0
	Number of Attack N-Grams Present	Calculated numeric value for strings If type of attribute is not string, value=0
	Number of Non-Attack N-Grams Present	Calculated numeric value for strings If type of attribute is not string, value=0
	Shannon Entropy of Attribute Value	Shannon entropy for strings If type of attribute is not string, value=0

Obtaining actual attacks for JSON instances was a challenge. Due to the lack of publicly available actual JSON attack data, it was necessary to simulate them. To aid in the simulation of attacks, a database of common attacks created by Muntner (2015) was used as a source. Unfortunately, Muntner (2015) did not have examples of "shellcode" or attack machine code that is directly executable by a system. Instead, shellcode for simulated attacks was generated using a security assessment tool, the Metasploit Framework Edition from Rapid7 (Metasploit Framework Edition, n.d.). These sorts of attacks are "templated attacks" in that they are fixed strings, but are randomly inserted in the attacked JSON data value. That they are randomly inserted in existing data is of importance. This means that detecting these attacks is not a simple matter of looking for a known value verbatim.

For attacks using some form of template, Table 5 shows the quantity of templates of each type that was available for this research. The Buffer Overflow attacks were available in much greater abundance than the other attacks since they could be programmatically generated in a wide variety of combinations. The potential experimental effect of the disparity of counts of templated attacks by type will be discussed in more detail in Chapter 4.

Table 5
Available Templated Attacks

Attack Type	**Attack Templates Available**
Buffer Overflow	2,659
JavaScript Insertion	78
Command Injection	1,276
SQL Injection	377

There are a number of decisions to be made in crafting attack data:

1. How often should JSON data instances be selected for attack insertion?
2. Within a JSON data instance, how are attributes selected for attack?
3. How are attributes selected for structural or value-tampering attacks?

4. How are value-tampering attacks made?

First, JSON data instances were selected at random, with a 5% to 95% probability of being attacked or not, depending on the iteration of the experiment. Second, based on a delimitation of this research, only one JSON attribute was attacked. Third, an attack type was selected from the following attack types:

- Buffer Overflow
- JavaScript/HTML Insertion Attack
- Command Injection
- SQL Injection Attack
- JSON Attribute Name Change
- JSON Attribute Added
- JSON Attribute Removed
- JSON Attribute Value Manipulation

Attacks made by modifying Boolean or number values are trivial since it is a straightforward value modification. Attacks on string attributes are much more difficult since they may contain considerably more complicated values. Fortunately, as mentioned previously, there are publicly available examples of attacks that have been used in a non-JSON context to compromise Web applications through normal HTTP means such as query parameter or request body tampering. These example attacks, which are available as or may be converted to strings of characters, were used in this research.

As mentioned previously, equal numbers of example attack types were not available. While an effort was made to secure adequate representation of each attack type, selection with replacement from the pool of attacks was made for attack insertion in the interests of providing adequate numbers of each attack type in both the training and test data. However, the same exact attack strings were not necessarily present in the both the training and test data.

The next consideration for attacking string attributes is how to insert these attacks in the JSON string data. Assuming one attack insertion per JSON attribute, the string could be inserted at the beginning, be inserted somewhere in the middle, be inserted at the end, or entirely replace the value. The approach for this research was to randomly select an attack string of the selected attack type and randomly select one of the insertion positions or to replace the value.

Figure 6 illustrates the way the training and test datasets were created. The source of the non-attack JSON data seen in Figure 6 was a Web browser capable of generating an HTTP Archive (HAR) file which is a record of all the interactions between the Web browser and Web server (Odvarko, Jain, & Davies, 2012). An example of this format may be seen in Appendix A. Coincidentally, the format of the HAR file is also JSON. The HAR file was filtered for any HTTP response that is marked as having a mime type of application/JSON, which indicates that the data is of JSON format (Bray, 2014). Appendix B offers a more detailed view of this flow.

The “holdout” method was used to split data into groups of 2/3 and 1/3, for the training set and test set, respectively (Han & Kamber, 2006). This split is detailed in Figure 6.

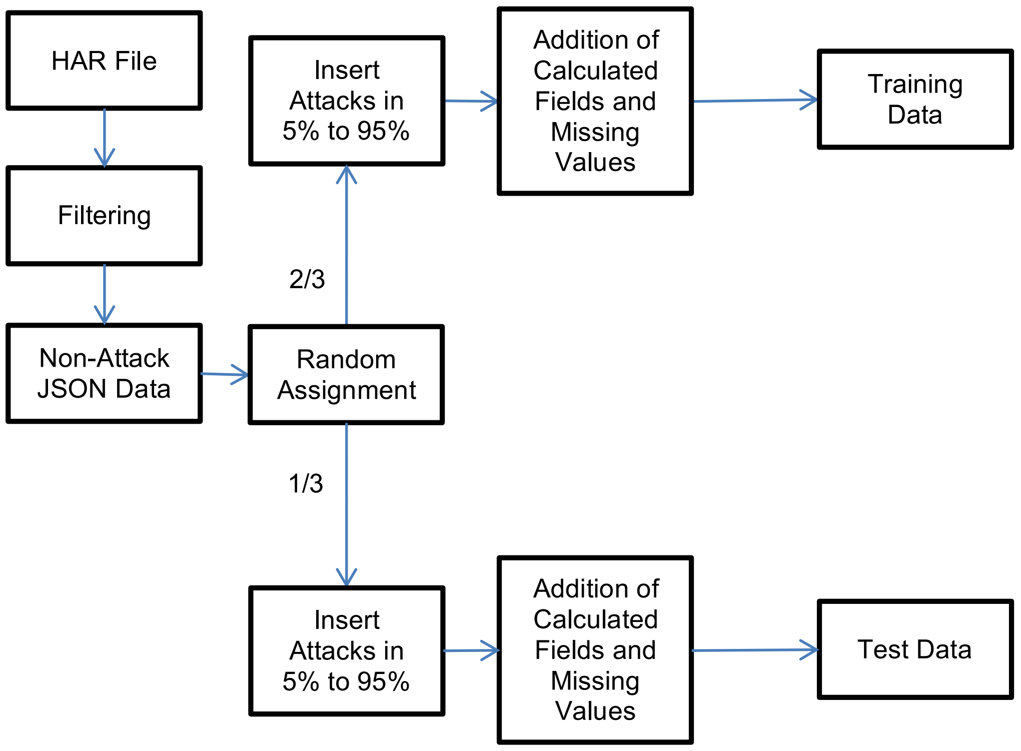

HAR File
Filtering
Non-Attack JSON Data
Random Assignment
2/3
1/3
Insert Attacks in 5% to 95%
Addition of Calculated Fields and Missing Values
Training Data
Insert Attacks in 5% to 95%
Addition of Calculated Fields and Missing Values
Test Data

Figure 6. Creation of Training and Test Data

After attack insertion and randomization, it was necessary to calculate additional columns of data such as n-gram counts, entropy, and string field length. Furthermore, since all JSON data structures in the dataset do not have identical fields, placeholders must be added for missing data values so that the Random Forest classifier can operate on all data instances in the dataset. Essentially what is being created is a superset of all fields and values. Median and mode values are used to replace missing values (Liu Peng, 2005). Note that when a missing value was replaced by a median or a mode, it was replaced by data collected from the same Web site. Here are the steps that were taken overall to prepare the test and training datasets:

- Missing values were replaced by the median value from the training set if the type of the value was numeric and the mode from the training set if the type of the value was

Boolean or string. As a convention, if the missing attribute is an object or array, the value was set to 1, as are all objects or arrays, present or not. The reason that this is done is that whether or not the attribute is missing is reflected in the value of the minimum Jaccard coefficient that has been calculated for this instance with respect to all non-attack instances in the training set for the same Web site. This provides a measure of the greatest similarity with respect to previously seen non-attack instances.

- Character n-gram lengths of 1, 2, 3, 4, and 5 were calculated.
- N-gram counts were determined by examining string data from the test set with the n-grams found in the training set for both attack and non-attack cases. By Assumption 16, n-gram counts were made with respect to the same attribute within instances from the same web site.
- As a convention, if a JSON attribute was found associated with different types, then the attribute was converted to a string type. For example, if the same attribute was seen with the numeric value 1234 and string value "abc," then the numeric value was converted to a string, i.e. "1234."

As this research was being planned there was concern that higher n-gram lengths would be computationally prohibitive. As an experiment, the average time in milliseconds necessary to calculate the n-grams in a random block of text was calculated over a range of block sizes from 100 to 2,000 characters and for n-grams in the range of 1 to 20 characters for a total of 180 combinations and 1,000 trials per combination. This used the same implementation as would be used in the full experiment for the research. This yielded the following regression equation which specifically excluded quadratic or higher terms as well as interaction terms for the sake of simplicity. The following regression equation with an R^2 value of .792974 was calculated. Details of the regression equation can be seen in Appendix C.

$$\begin{aligned} &time\_in\_miliseconds \\ &\quad = -1.50712 + 0.00128497 * number\_characters \\ &\quad + 0.141355 * ngram\_length \end{aligned} \tag{4}$$

As can be seen in the equation, the *ngram_length* parameter contributes .14 milliseconds each time the n-gram count is increased by 1 character. As a simple example, neglecting the effect of the number of characters over which the n-grams were being calculated, 10 n-grams would take about 1.4 milliseconds. Conversely, 1,000 characters will contribute about 1.3 milliseconds. Consequently, on a unit for unit basis, adding an additional character to the n-grams which are being calculated has a much more dramatic effect than adding another character to the block of text over which the n-grams are being calculated. This means that controlling the length of the n-grams being calculated is of greater importance in minimizing the time spent. With the selected value of the length of n-grams being 5, then the regression simplifies to:

$$\begin{aligned} &time\_in\_miliseconds \\ &\quad = -0.800345 + 0.00128497 * number\_characters \end{aligned} \tag{5}$$

Neglecting violation of causality for $0.00128497 * number\_characters$ being less than 0.800345, calculating the n-grams (n=5) of 1,000 characters would take about $-0.800345 + 0.00128497 * 1{,}000 =$ 0.484625 milliseconds or about half a millisecond. While this is only calculating the time spent in calculating n-grams, rather than counting the number of matching n-grams between two blocks of text, as is done as part of the classification process, a value of n=5 provides acceptable running times for this research.

Also, as this research was initially contemplated, a dimensionality reduction step was potentially planned to occur before the training or test data was used. This was for two reasons. First, since the number of unique JSON attributes might be quite large, it was feared that the time for computation necessary to both train and use the classifier ensemble would be excessive. Dimensionality reduction would reduce the number of attributes to a more manageable size. Second, dimensionality reduction could have been used as a way of discovering which attributes were most predictive of attacks. In the end, both of these points were otherwise addressed. In actual practice the random forest algorithm implementation was able to perform the necessary computations within reasonable time bounds. So from a purely speed perspective, dimensionality reduction was not necessary. In addition, the implementation of the random forest algorithm used provided a feature ranking as a part of its implementation and in a more straightforward way than would have been possible with dimensionality reduction techniques. Consequently, dimensionality reduction was dropped from the experimental procedure. It is possible that the speed of training and classification may have been increased by dimensionality reduction techniques. However, optimization of the classification process was not a specific goal of this research and has been left for later research efforts.

Data Collection

The data collected consisted of the outputs of the classifiers from both the experimental group and the control group for each dataset. This process was repeated for all experimental permutations of training attack percentages, test attack percentages, and numbers of classifiers. Also, the training and test data sets, with information about whether an attack was present and what kind of attack, served as inputs for analysis.

Data Analysis

The analysis process was repeated for the control and experimental groups for all data sets. Since it is also desirable to have an overall result, the approach taken in this research was to sum the entries in the confusion matrix in Table 3 across all data sets and a given classifier group. After summing the confusion matrices, the χ^2 statistic was calculated and evaluated as described for the case of a single dataset.

It was also informative to compare the different trials of the experiment since it was repeated for different percentages of attack instances in the 5% to 95%. During the planning phase of this research, the upper bound of attacks was set at 50%. However, this was expanded to 95% in the actual research. The 50% level was set due to concerns about computational limits. In practice, these concerns proved unfounded, so the upper range was extended to 95%. This had the added practical benefit in that it would also test the classifiers under the load of a very aggressive attack. And, as will be seen in Chapter 4, interesting features of classifier performance would have been missed with the original 50% upper bound.

A tool for comparing binary classifiers known as the Receiver Operating Characteristic (ROC) was selected to aid in classifier comparison for different percentages of attack prevalence (Kolo, 2011). A hypothetical ROC curve or plot that might result from the series of experiments may be seen in Figure 7. The diagonal line represents a purely random classifier. The isolated points are placed at (x,y) coordinates in which the x is the False Positive Rate (FPR), or the ratio of negatives that are incorrectly labeled as positives to all negatives (i.e., False Positives divided by negatives), and the y coordinate is the True Positive Rate (TPR), or the ratio of correctly identified positives to all positives (i.e., True Positives divided by positives) for a given classifier (Han & Kamber, 2006). This allows a rapid visual comparison of the classifiers that were trained and tested with different prevalence of attack instances in the training and test sets. Furthermore, any of the points that fall above the random chance diagonal line exhibited better results than pure chance expectation. Whether or not a given classifier is performing better than chance expectation can be inferred from the MCC. However, the ROC curve provides a simple visual indication of this for all classifiers.

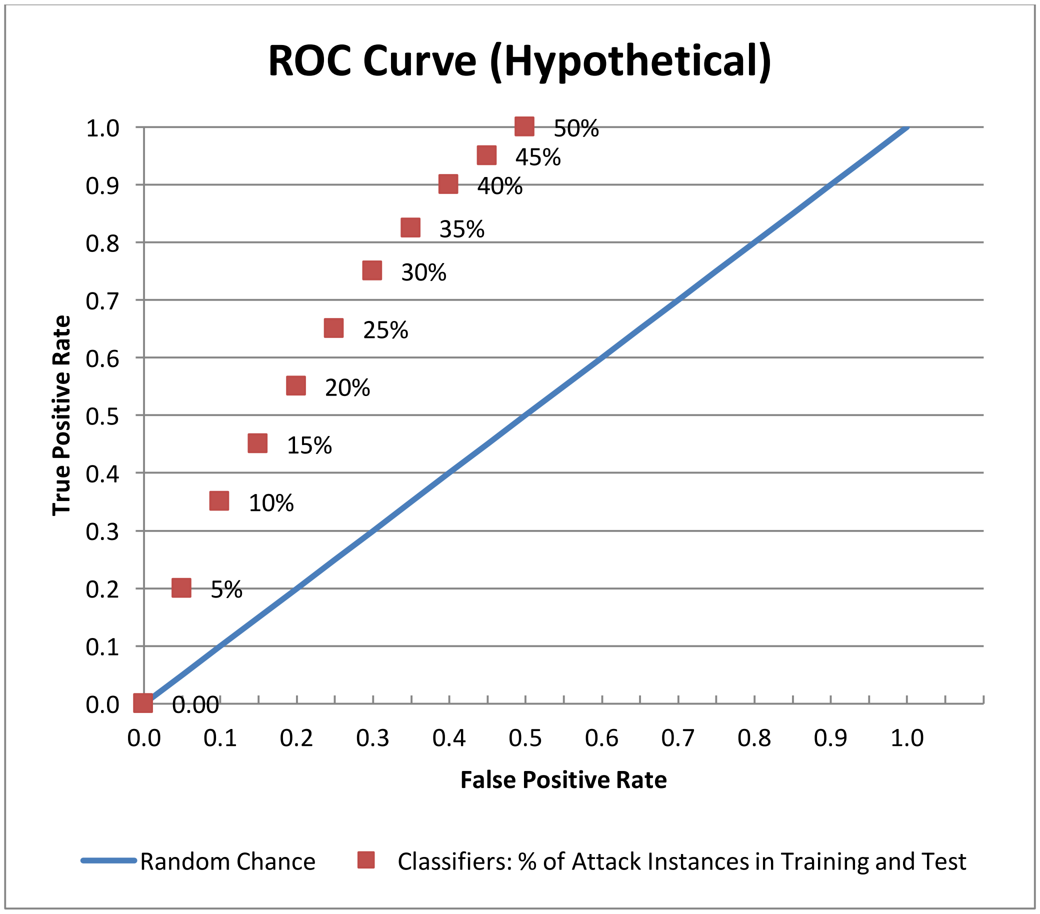


Figure 7. Hypothetical ROC Curve

Tools Utilized in the Research

A number of free and commercial tools were used for this research. HAR archive files were collected using the Mozilla Firefox Developer Edition (Firefox Developer Edition, n.d.). Buffer overflow attacks were simulated using the Metasploit Framework Edition from Rapid7 (Metasploit Framework Edition, n.d.). The JetBrains PyCharm Community Edition Python development environment was used with Python 2.7 in the development of all the software used in the research (PyCharm Community Edition, 2015). Software created for this research made use of the Python Scikit-learn machine learning library (Pedregosa et al., 2011). In addition, the commercial statistics tool Statgraphics Centurion XVI was used, in particular for regression analysis as well as general data analysis (Statgraphics Centurion XVI, 2012). The Python Statsmodels library was also used for regression analysis (Seabold & Perktold, 2010). The Python Pandas data analysis libraries were used in general analysis (McKinney, 2010). Python Matplotlib library was used for most of the data visualizations (Hunter, 2007). The Numpy Python library was used for direct calculations or with other libraries (van der Walt, Colbert, & Varoquaux, 2011). Statistics functions were used from the Python SciPy library (Jones, Oliphant, Peterson, & others, 2001). The PyPy Just In Time Compiler (JIT) for Python implementation was used for dramatic speed increases in data preparation and attack simulation phases (Lutz, 2013; "PyPy - What is PyPy?," n.d.). In addition, the Python SymPy library was used in Chapter 4 for verifying manual calculations involving symbolic manipulation and equation solving (SymPy Development Team, 2014).

Chapter 4: Results and Discussion

Overall Statistics

A total of 7,220 trials were conducted. These trials consisted of permutations of varying percentages of attacks in training, percentages of attacks in testing, and the number of classifiers used within the Random Forest ensemble. The attack percentages were varied between 5% and 95% in increments of 5%. The number of classifiers was varied between 1 and 95 classifiers in increments of 5 classifiers, except for the first and second values of 1 and 5 classifiers, respectively. The training phase of each trial had 1,765 instances of JSON objects. The test phase had 882 instances.

As noted in Assumption 15, the original intent was to have equal numbers of each type of attack, but in practice this was not possible. Specifically, JSON objects in the training and test datasets did not always have a structure vulnerable to a name change attack. JSON objects consisting of entirely unnamed lists of objects were present in the training and test datasets in sufficient quantities to prevent equal numbers of each attack type. These instances had no JSON attribute names to change, so they were not susceptible to the JSON name change attack. In this case, another attack type was randomly selected. Still, experimental trials included 241,640 occurrences of the JSON Attribute Name Change attack, as can be seen in Table 6, in which attack counts for the other attack types may be seen as well. All other attack types occurred in roughly equal proportions. Per Assumption 15, this should not affect the validity of the resulting attack mix.

Table 6
Prevalence of Attack Type

Attack Type	Number of Attacks of this Type	Percentage of Overall Attacks
Buffer Overflow	418,100	13.15%
JavaScript Insertion	420,660	13.23%
Command Injection	421,820	13.26%
SQL Injection	419,320	13.18%
JSON Attribute Name Change	241,640	7.60%
JSON Attribute Added	420,800	13.23%
JSON Attribute Removed	419,440	13.19%
JSON Attribute Manipulation	418,820	13.17%

Randomness of Control Group Classifiers

An objective of this research was to show that a trained classifier performs better than an untrained or “random” classifier. In principle, an untrained binary classifier should perform no better than a “coin flip” or chance expectation. An important part of validating the experimental approach is to verify that the random classifier which was used is indeed random. A random number generator is used in this research as a random binary classifier to produce ones and zeros, which represent “true” or an attack is present and “false” or an attack is not present. This random number generator should produce roughly equal numbers of zeros and ones. As an experiment, the random number generator used in this research was made to generate 500 sets of yes/no decisions, with the sets ranging in size from 10 to 2,500 decisions in increments of 5 between sets. The resulting scatter plot of the number of zeros and number of ones in each trial is shown in Figure 8. As can be seen, the numbers of true values (represented by 1’s) and false values (represented by 0’s) all fall roughly along the diagonal, as would be expected if equal numbers of each were present.

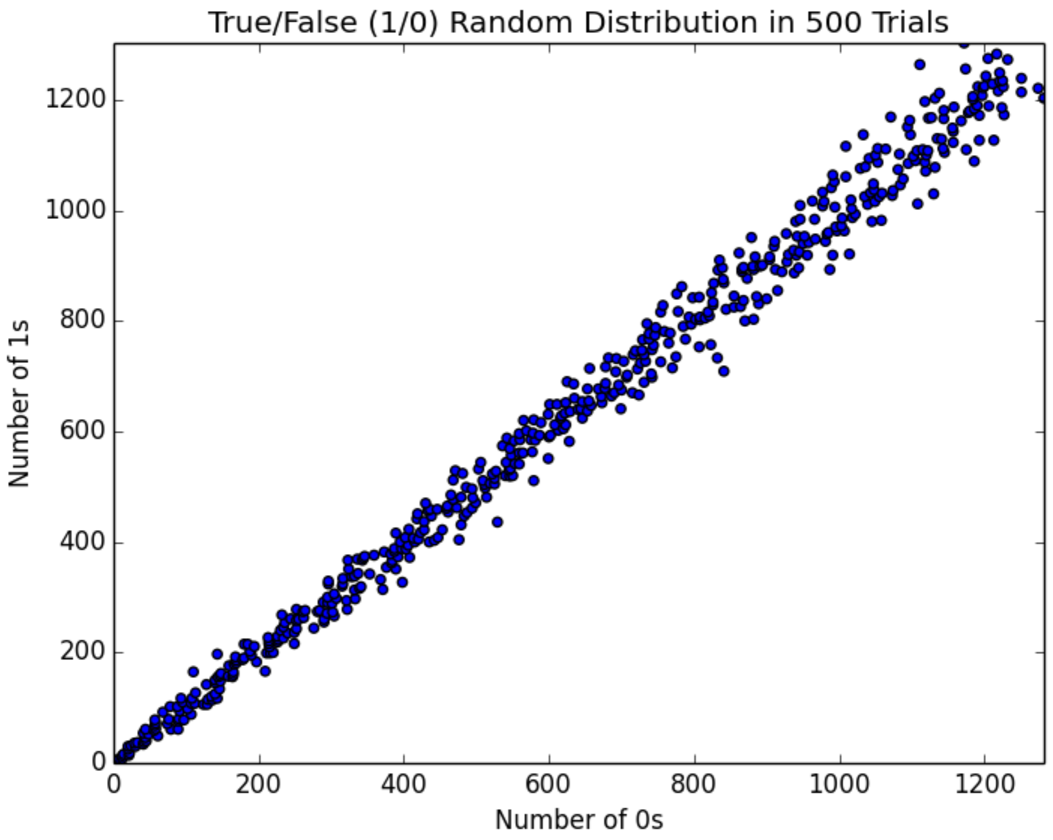


Figure 8. Random Number Generator Performance

More formally, the "Monobit" test can be used to evaluate randomness, at least in the sense that there are equal proportions of a binary outcome (Rukhin et al., 2010). The Monobit test calculates a p-value that is less than 0.01 if a binary sequence is not random. The same trials that were shown in Figure 8 are repeated in Figure 9, but with the Monobit p-value output as the y-axis. A dashed line indicates the critical 0.01 p-value. The vast majority of points is substantially above or greater than the 0.01 p-value cutoff. This supports the assumption that the random number generator used in this research is sufficiently random.

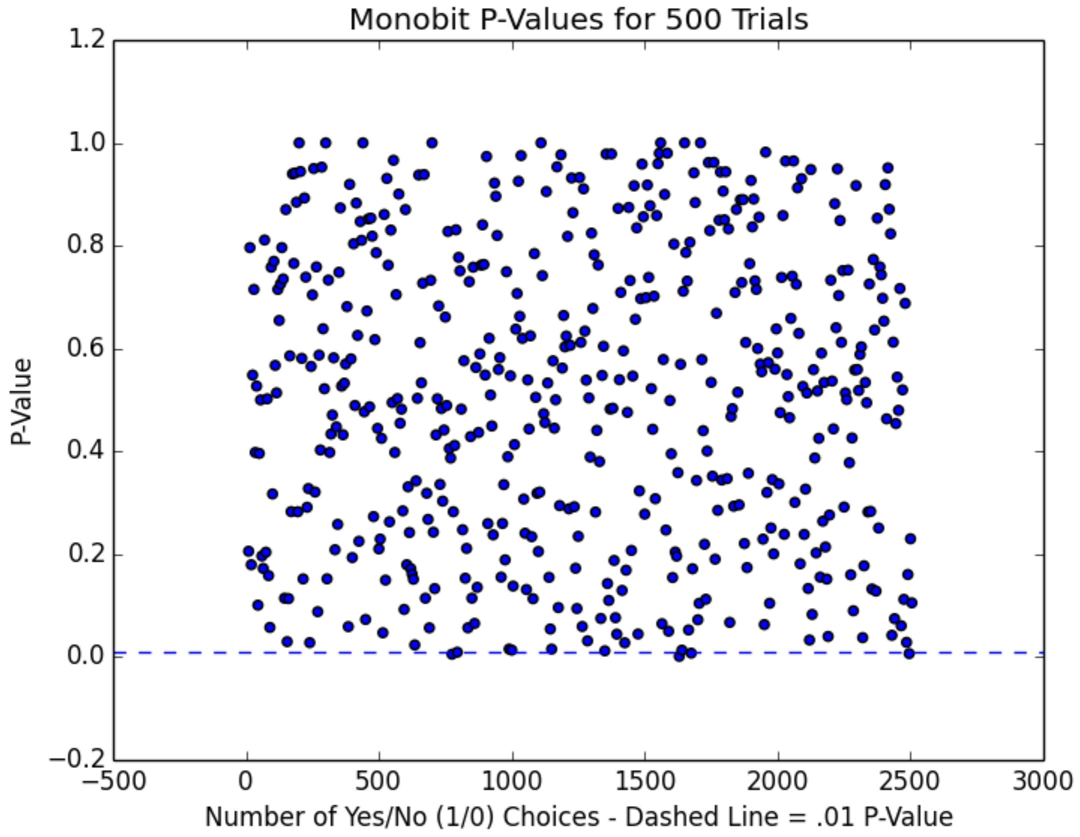


Figure 9. Monobit P-Values

So far random inputs to the classifier have been considered. It is also worthwhile to examine the output of a random classifier to verify that random inputs produce random outputs. The ROC chart can aid in determining whether or not the classifier is behaving randomly. In Figure 10, the ROC chart may be seen for the control group pretest. A small amount of "jitter" or random x and y axis perturbation has been added so that the points may be better visualized. All points fall along the diagonal, as would be expected for a random classification in an ROC plot. An additional feature to this plot is that the MCC value is indicated by color. The MCC is a more holistic metric which also takes into account the false negative and true negative rates as well as the true positive and false positive rates. Consequently, a point with a high true positive rate and a low false positive rate that is seen as a higher performance point by ROC analysis may have a lower-scoring performance by the MCC metric. Plots for the posttest curve and the pretest experimental group, not shown here, are virtually identical as they are all produced by the same type of random number generator.

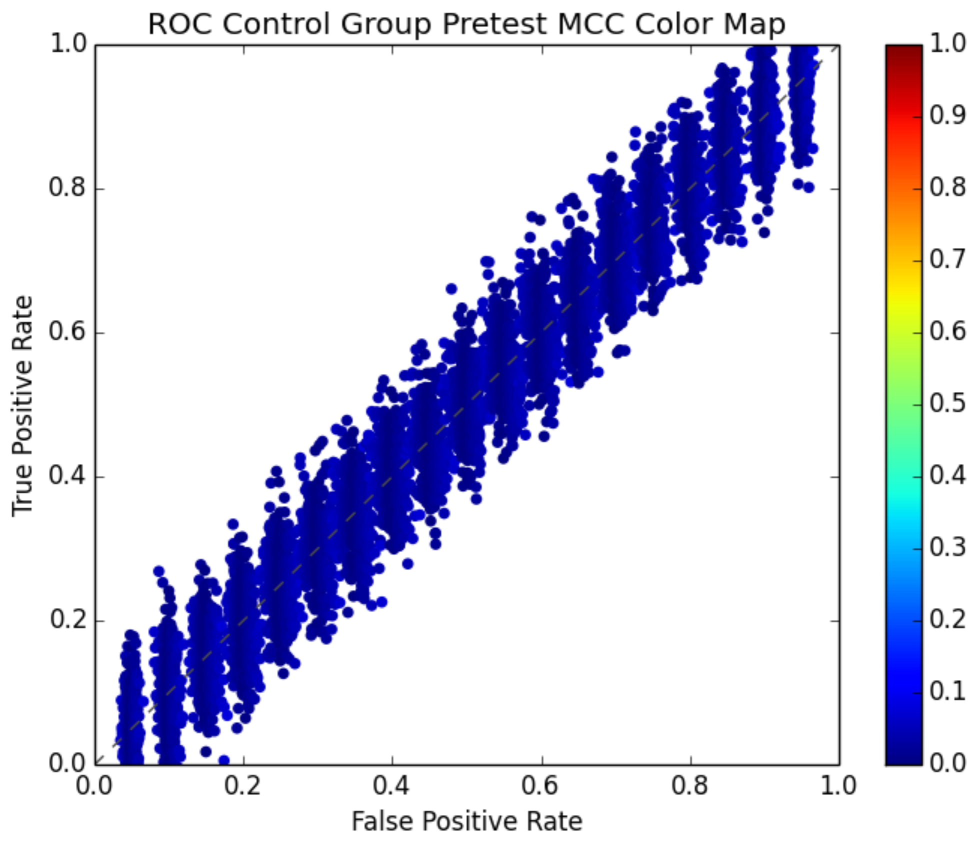


Figure 10. ROC Curve for Control Group Pretest

Another way of evaluating the randomness of the random classifier is to create box plots for the pretest and posttest MCC values of the control group. This is shown in Figure 11. The values for this boxplot are taken from the cumulative runs of MCC values from all experimental trials. The plot displays the threshold for the MCC to be statistically significant as a dashed line. In this plot, that value is approximately ±0.0467. As was previously stated, per Baldi et al. (2000), $\chi^2 = N * MCC^2$. Refactoring for MCC gives

$$MCC = \sqrt[2]{\chi^2/N} \tag{6}$$

The χ^2 value is selected that corresponds to a p-value of .05 for one degree of freedom or $\chi^2 = 3.8415$. In this case, N= 1,765, so MCC = $\sqrt[2]{\chi^2/N} = \sqrt[2]{\frac{3.8415}{1765}}$ = 0.0467. However, since the MCC term is squared in $\chi^2 = N * MCC^2$, MCC can be positive or negative. Therefore, MCC= ±0.0467. This is the method used to subsequently calculate any critical MCC values, though the value of N will vary. Anything outside of this range is considered to be statistically significant at the p=.05 level. Therefore, outputs of a random classifier should fall inside this range. Points do appear in this diagram beyond the horizontal lines or "whiskers" used to indicate the range of Quartile 1 - 1.5 times the Inter Quartile Range (IQR) to Quartile 3 + 1.5 times the IQR. By convention, values outside of this range are considered outliers (Han & Kamber, 2006). By definition 50% of observations will fall within the "box" of the box and whiskers plot. The placement of the MCC threshold between these two features of the box and whiskers plot place it at the p=.05, beyond which there is only a 5% chance that a data point is statistically significant. For the pretest, there are 359 values that have p-values less than 0.05. This is out of 7,220 values, or about 4.97%. For the post-test there are 245 values out of 7,220 or about 3.39% that have p-values less than 0.05. In both cases, these numbers support that less than 5% of such extreme values would occur at the p=.05 significance level. Another noteworthy observation is that in this diagram the statistical parameters of the pretest and posttest classifiers are essentially the same, as would be expected if a classifier were untrained.

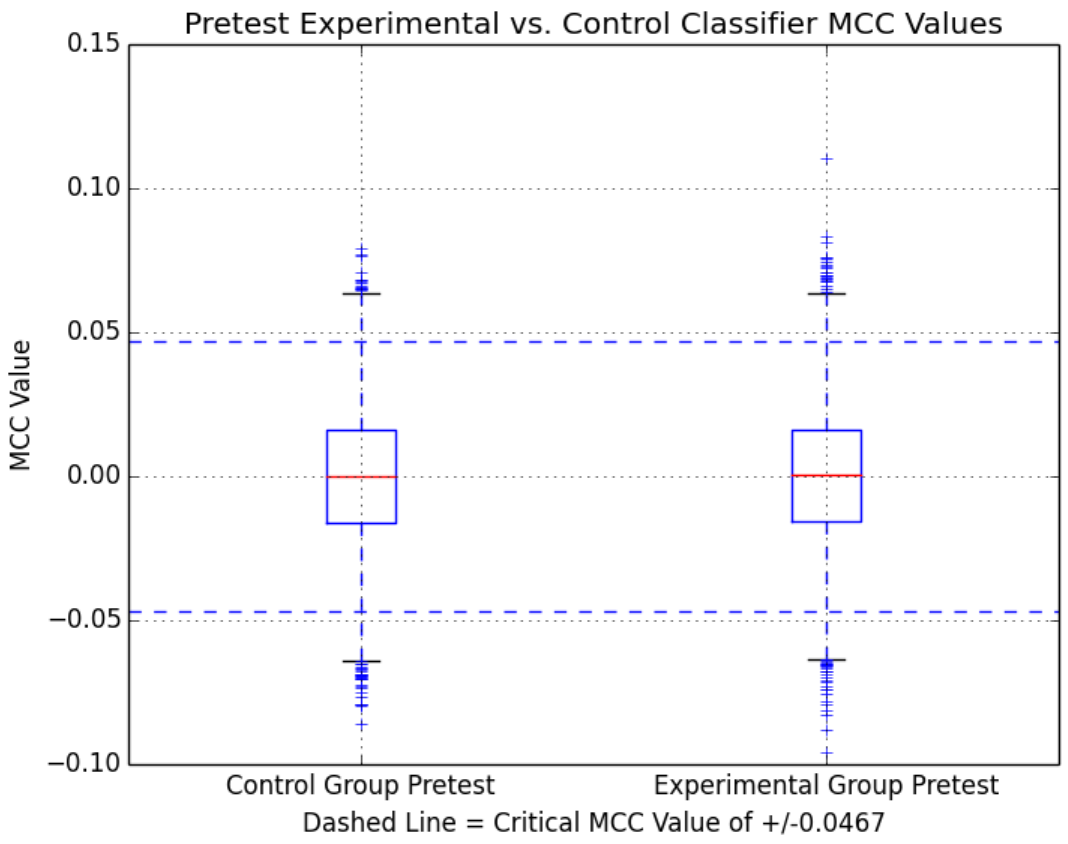


Figure 11. Pretest Boxplot for Control and Experimental Groups

Overall Performance by MCC

This research intends to show that a trained classifier performs better than a random classifier for detecting attacks. It has already been shown that the random classifier is sufficiently random. Therefore, it is now possible to compare the performance of a trained classifier to that of an untrained or random classifier. Again, the boxplot may be employed. Figure 12 shows such a comparison. The values for this boxplot are taken from the cumulative runs of MCC values from all experimental trials. Dashed lines have been added to indicate the critical MCC values, in this case ± 0.0660. This value is given by substitution into Equation 6 as MCC = $\sqrt[2]{\chi^2/N} = \sqrt[2]{\frac{3.8415}{882}}$ = ± 0.0660. Again, with the exception of outliers, the untrained random classifier values predominantly fall within the range which is not considered to be statistically significant. On the other hand, the trained classifier, with the exception of a few outliers, falls within the range which is considered to be statistically significant.

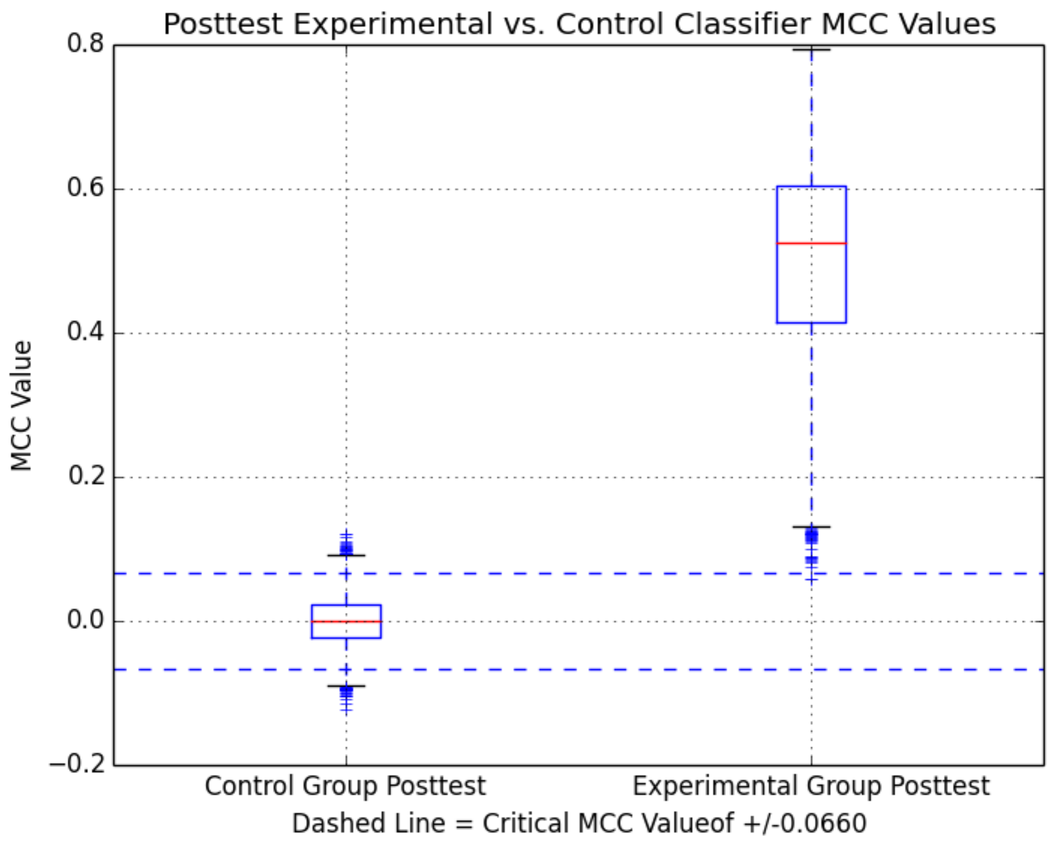


Figure 12. Posttest Boxplot for Control and Experimental Groups

Another way of examining the results is to look at the calculated p-values for the classifiers in each trial. Statistically significant classifier performance will have a p-value less than .05. Of 7,220 experimental trials, 7,218 had a p-value less than .05. Only two trials had a p-value greater than or equal to .05. The next largest p-value was 0.02772, which is below the .05 cutoff. The two p-values greater than .05 can safely be considered outliers.

Therefore, hypothesis H_0 fails to be supported, and H_1 fails to be rejected, so the broad question of whether trained classifiers can outperform random classifiers is answered. Furthermore, the Experimental Group Posttest box indicating 50% of the values falls between approximately .4 and .6, which are higher values of MCC.

On the other hand, a classifier that does just above chance expectation may only be marginally useful. Fortunately, it is also possible to separate good classifiers from those which are only able to exceed chance expectation. By Powers (2011), the MCC can be treated as equivalent to Pearson's Correlation Coefficient. Table 7 was produced by taking ranges of values and qualitative descriptions of the Pearson correlation coefficient from Evans (1996) and associating them with the number of trials that produced MCC values within these ranges. By this measure, there was a moderate or better strength for 77.87% of all trials.

Table 7
MCC Values Versus Quantitative/Qualitative Ranges

Strength	**Range**	**Number in this Range Out of 7,220 Trials**
Very Strong	.80 to 1.0	0
Strong	.60 to .79	1,921
Moderate	.40 to .59	3,413
Weak	.20 to .39	1,374
Very Weak	0 to .19	142

It is also informative to merge all the proceeding boxplots into one plot to see performance of both control group and the experimental group side by side, both pretest and posttest. Since the critical MCC value varies due to the number of trials in the pretest versus posttest, it will not be added as a dashed line as was done previously. The composite plot can be seen in Figure 13.

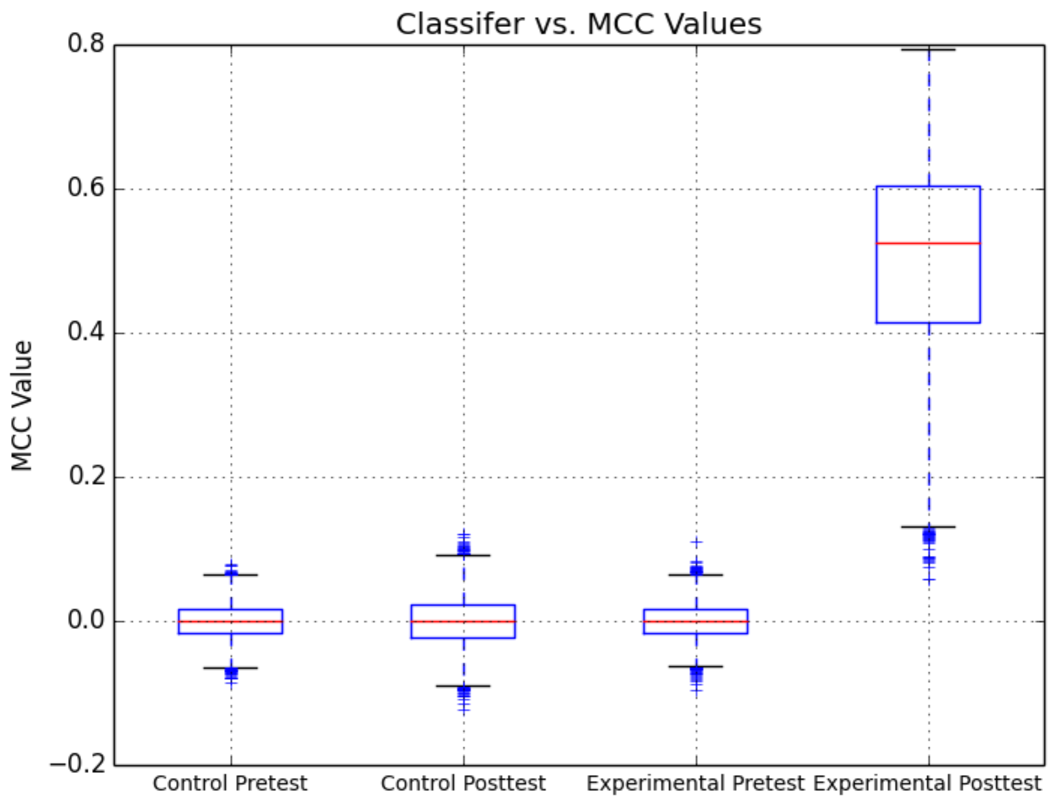


Figure 13. Pretest/Posttest Boxplot for Control and Experimental Groups

A detail that stands out is that the Control posttest box is larger than the Control Pretest and Experimental Pretest boxes. These are all random classifiers, so why are the statistical parameters different? The reason can be found in the number of yes/no classifications that are made in the pretest groups versus the posttest groups. The pretest groups are larger, while the posttest groups are smaller. This leads to the reasonable speculation that this is due to the influence of more extreme points in a small population causing a larger variance. To verify this, an experiment was conducted in which 10,000 trials of varying instance sizes of random yes/no (1/0) decisions were made. The MCC value was calculated for each trial. The results can be seen in Figure 14. The number of instances is the number of binary decisions in each trial. The predicted effect is clearly shown. As the number of instances in the group increases, the box of the boxplot becomes smaller. More formally, this can be seen as an example of the Law of Large Numbers, which states that the larger the number of samples, the greater the tendency towards the actual population mean and variance (Grinstead & Snell, 1997).

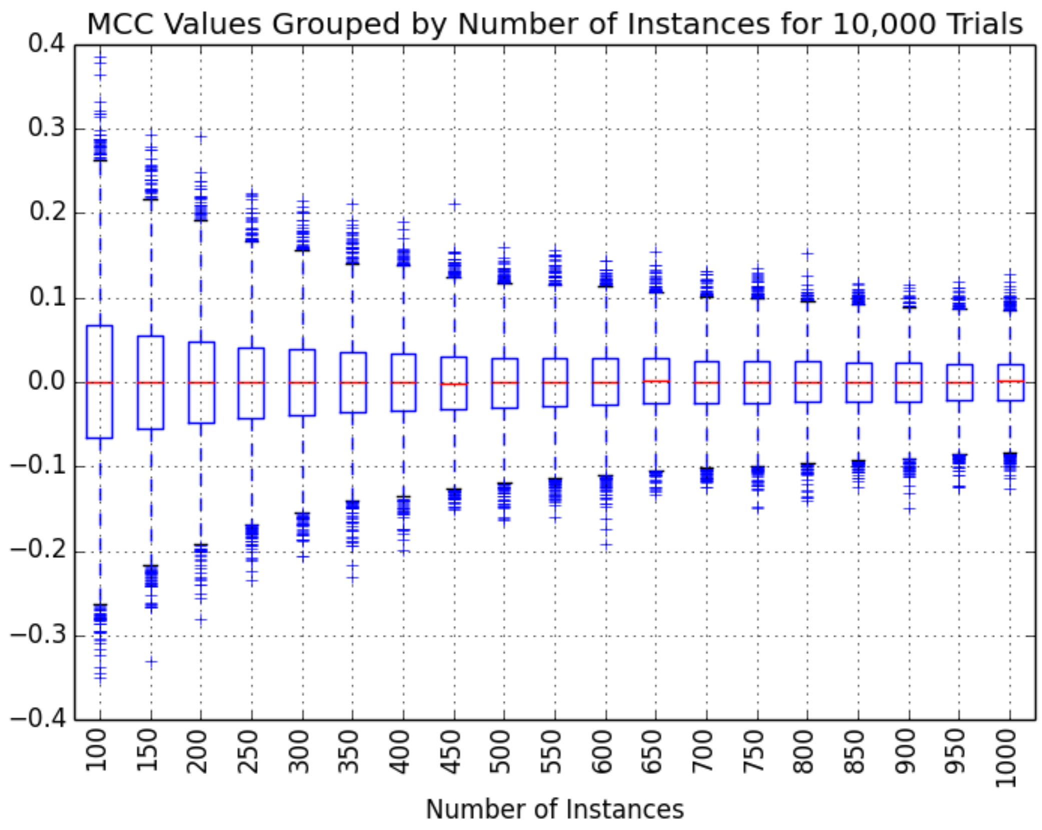


Figure 14. Boxplot of MCC Values Versus Numbers of Instances

MCC as a Function of Attack Percentages and Number of Classifiers

So it has been shown that a trained classifier can be substantially better than an untrained or random classifier. But the next question that arises is if there are specific parameters that affect classifier performance. In the trials for this research, different permutations of three main parameters were varied: percentage of attacks in the training set, percentage of attacks in the test set, and the number of classifiers in a Random Forest instance. A question that arises is whether the attack type should be considered as a 4th parameter. The assumption for this research is that multiple different attack types will occur in the environment in which the classifier operates (Assumption 13). As such, training and testing for a specific attack type at a time may not mirror reality. However, a different approach than the one taken in this research could involve an ensemble of classifiers in which each classifier is trained to a specific attack. This alternate approach is not addressed in this research, though it is mentioned as a possible future research topic in Chapter 5. Nevertheless, the performance of the current approach by attack type will be evaluated later in this chapter.

If the relationship between the parameters and classifier performance were known, it would aid in tuning a classifier for the best performance for a given application. ROC plots can help as an initial survey of how these parameters may affect performance. Three ROC plots modified to use coloration to show a third parameter are shown. The first can be seen in Figure 15.

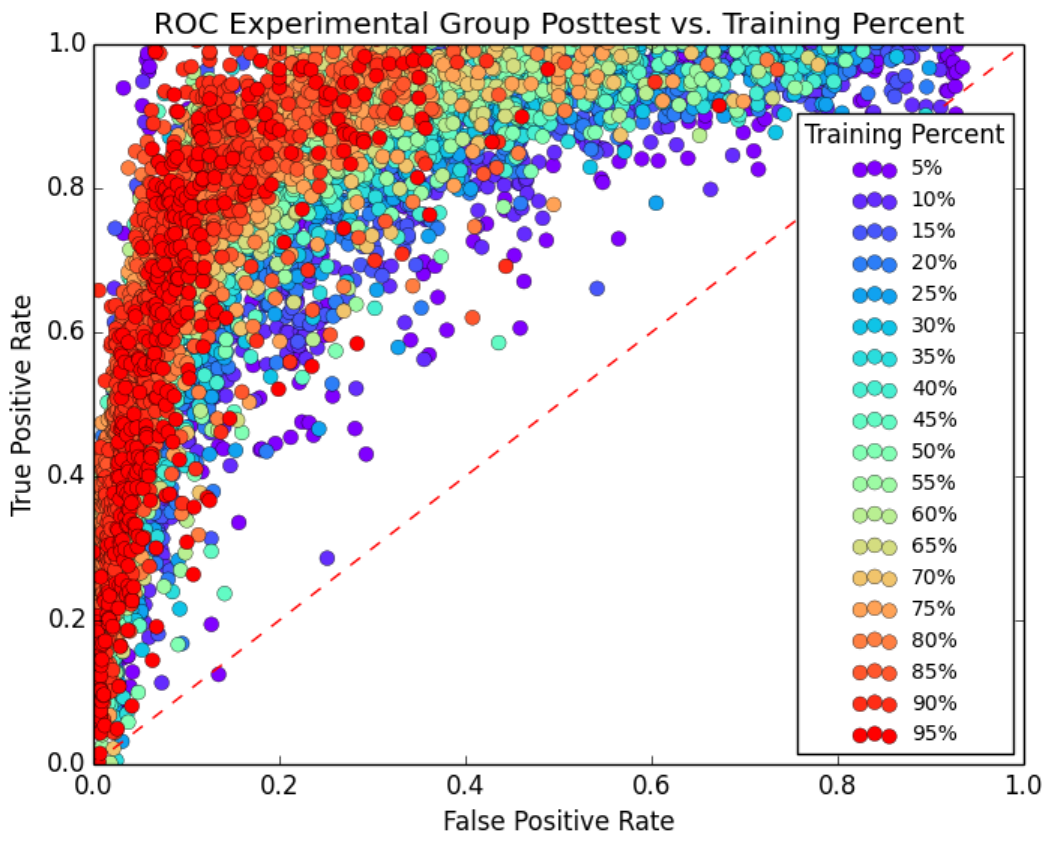


Figure 15. ROC Plot for Training Percentages

In this curve the percentage of attacks in the training set is indicated by the color of the point. To improve the ability to visually separate overlapping points, “jitter” has been added. Higher training percentage values can be seen clustering around the vertical axis with an increase in the False Positive Rate direction starting at a True Positive Rate of about .5. Points in this region closer to the y-axis will have a high True Positive and low False Positive Rate. This means that these points would correspond to classifiers that correctly identified a large number of true attacks, while at the same time not misidentifying normal instances as attacks. This would seem to suggest that the more attacks to which a classifier is exposed to in training, the better its measured performance, at least as exhibited by an ROC plot.

The next ROC plot with coloration of points indicating the percentage of attack instances in the test dataset can be seen in Figure 16. Again, “jitter” has been applied for better visualization. An interesting relationship to note here is that points that are associated with a higher percentage of test attacks are found higher on the y-axis. That is, they have a higher True Positive Rate (TPR). However, these points may be found across the width of the entire ROC plot. This means that there is not a particularly strong relation of these points to a high or low False Positive Rate (FPR). Therefore, a point being associated with a higher percentage of test attacks does not necessarily mean that it is a better classifier in all measures. But the association of higher percentages of test attacks with a higher True Positive Rate (TPR) does mean that these classifiers will appear to better identify real attacks when they are subjected to a large number of attacks during the testing phase. This should not be construed to mean that a classifier can be improved by testing it since any exposure to attacks after the training phase will simply reveal the underlying ability of the classifier as trained. One possible explanation is again the Law of Large Numbers (Grinstead & Snell, 1997). If only a small number of attacks is present in the test dataset, then it may be that the classifier performance is over or understated with respect to the actual performance values. A test dataset with a larger percentage of attacks may simply be more accurate in revealing the true performance measures. On the other hand, a test dataset with few non-attack instances will be more likely to distort measures such as the True Negative Rate (TNR) since there would be very few non-attack instances available for the calculation of classifier metrics.

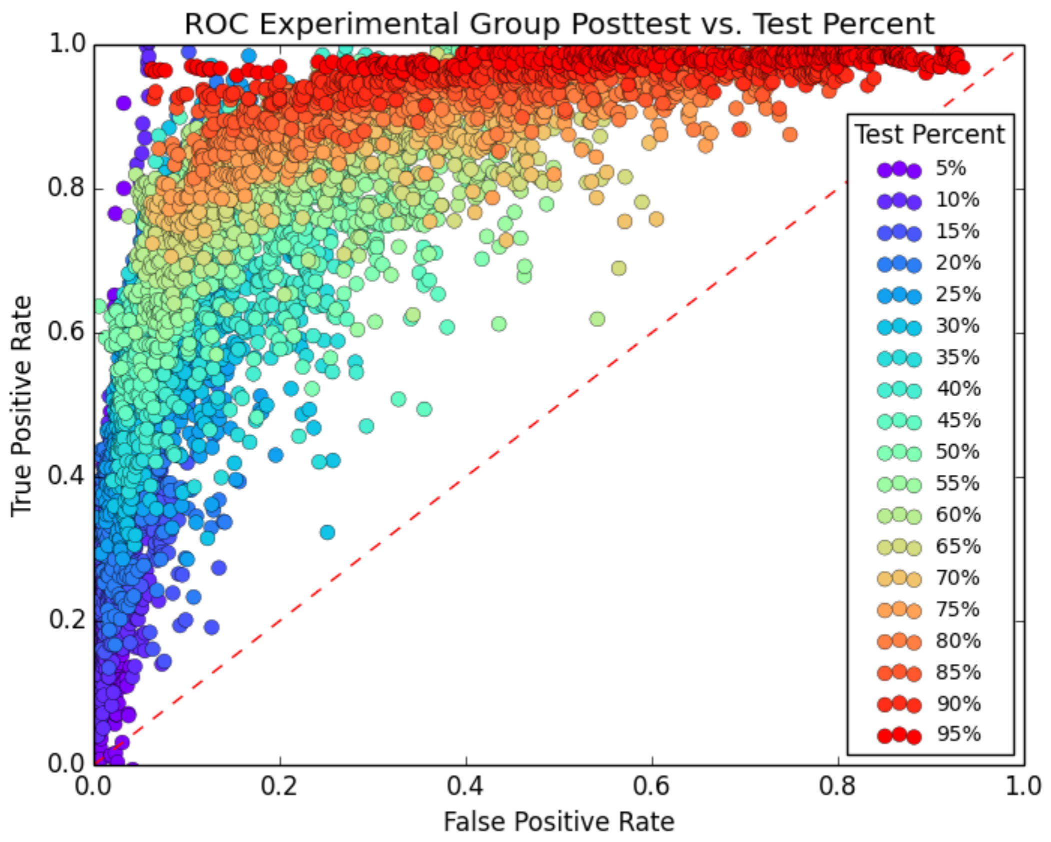


Figure 16. ROC Plot for Test Percentages

As mentioned previously, a point with a high MCC value may or may not appear as a high performing point on the ROC curve. The following ROC plot in Figure 17 shows MCC values via point color with respect to ROC curve placement. Higher MCC values correspond with the regions showing high percentages of training attacks in Figure 15, though only at the points that occur with a high True Positive Rate. More intensely red and orange colors indicating higher MCC values may be found in the upper left-hand corner of the plot. This would be expected in an ROC plot as the upper left-hand corner is the location of points with the highest True Positive Rate (TPR) and the lowest False Positive Rate (FPR). But it is important to note that some of the points in the plot with lower MCC values are higher on the y-axis and farther to the left on the x-axis with respect to points with higher MCC values. In other words, some points representing classifiers with a high MCC value do not fall on areas in the ROC curve that indicate the highest performance based on the True Positive Rate (TPR) and the False Positive Rate (FPR). This illustrates some of the shortcomings in relying purely on the ROC plot.

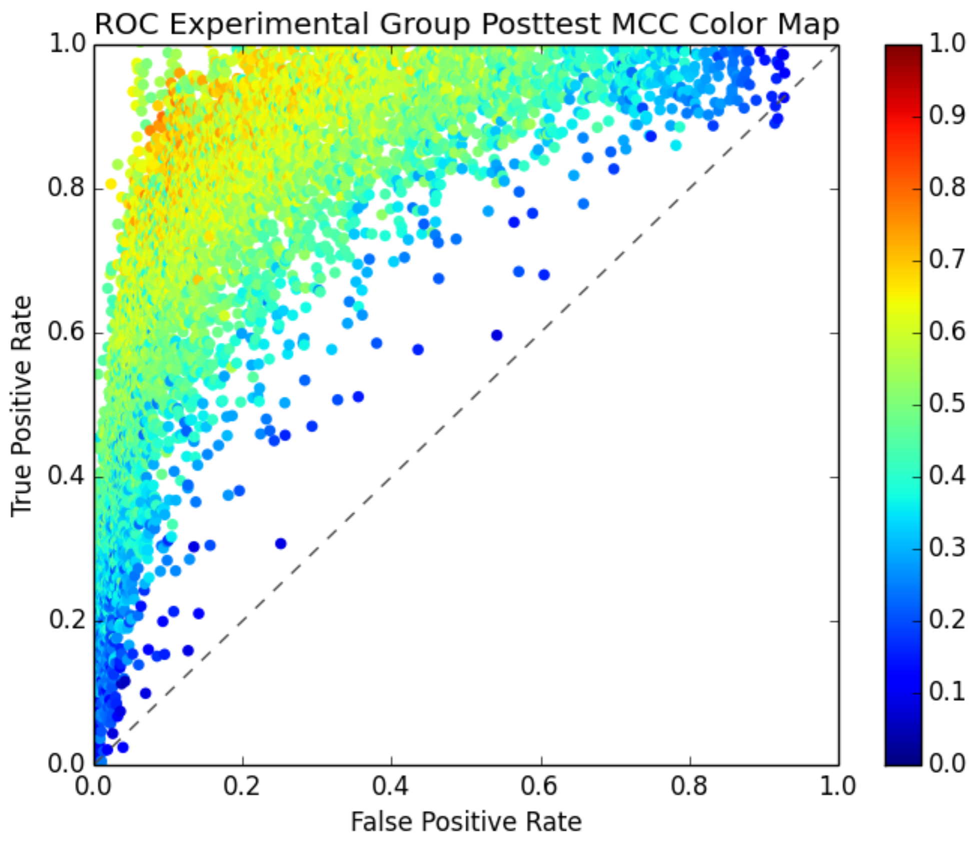


Figure 17. ROC Curve Showing MCC Values

The next step in analysis is to create a model in order to understand the influence of the independent variables as well as to aid in their selection for a desired set of design parameters. Multiple linear regression can be employed to find useful information about the relationship between independent and dependent variables even when the relationship is not linear (Draper & Smith, 1998). Furthermore, by Gupta & Guttman (2013), even a polynomial regression model may be treated as a multinomial linear regression model by appropriate substitution of independent variables. Therefore, a multiple linear regression model is an appropriate candidate to attempt to model the relationship between classifier parameters and MCC values. A model was built using the percent of attacks used for training, the percent of attacks used for testing, and the number of classifiers as independent variables, and the trained classifier MCC value as the dependent variable. The regression line was built by randomizing the order of the results and taking two-thirds to build the regression line, with one-third held back for validation. The initial model did not include quadratic or interaction terms. The best model of this form had an R^2 value that only accounted for 13.9613% of the variability in the dependent variable. Quadratic and interactions terms were added, with much better results. Additionally, following techniques from Draper & Smith (1998), a logarithmic transformation of the dependent variable was performed to make the residual plot more regular and Studentized residuals greater than 3 were removed. This achieved an R^2 value of .762001 against the two-thirds of the result set used to build the regression line. Against the validation set this yielded an R^2 value of .699459. This increases the likelihood that the regression model will apply to other datasets and not just the one used to build it. The R^2 value is respectable, but not so high a value that would indicate potential overfitting. All terms are statistically significant at the p=0.05 level. Appendix D lists the Statgraphics output from the creation of the regression equation. Appendix E illustrates plots of the regression equation and residuals. The final regression equation was as follows:

$$
\begin{aligned}
&ln(exp\_posttest\_mcc) \\
&\quad = -1.37911 + 0.0156883 * training\_percent\_attacks \\
&\quad + 0.0163151 * test\_percent\_attacks + 0.00483186 \\
&\quad * number\_classifiers - 0.000222984 \\
&\quad * training\_percent\_attacks^2 - 0.000203297 \\
&\quad * test\_percent\_attacks^2 - 0.0000426789 * number\_classifiers^2 \\
&\quad + 0.000132323 * training\_percent\_attacks \\
&\quad * test\_percent\_attacks + 0.00000783217 * test\_percent\_attacks \\
&\quad * number\_classifiers
\end{aligned} \tag{7}
$$

Judging by the magnitude of the coefficients of the terms of the equation, percentage of attacks in the training dataset and test sets will have considerable influence on the MCC value. The number of classifiers will also have somewhat lesser influence. But there are also other quadratic and interaction terms that are statistically significant. A visualization of the regression line can help. This can be seen in Figure 18. The MCC value is the y-axis and each point on the x-axis is a tuple of training attack percent, test attack percent, and number of classifiers. The x-axis tuple parameters cycle through their values in order. This compares favorably with the actual experimental data, as can be seen in Figure 19 in which the regression line is overlaid on the experimental data.

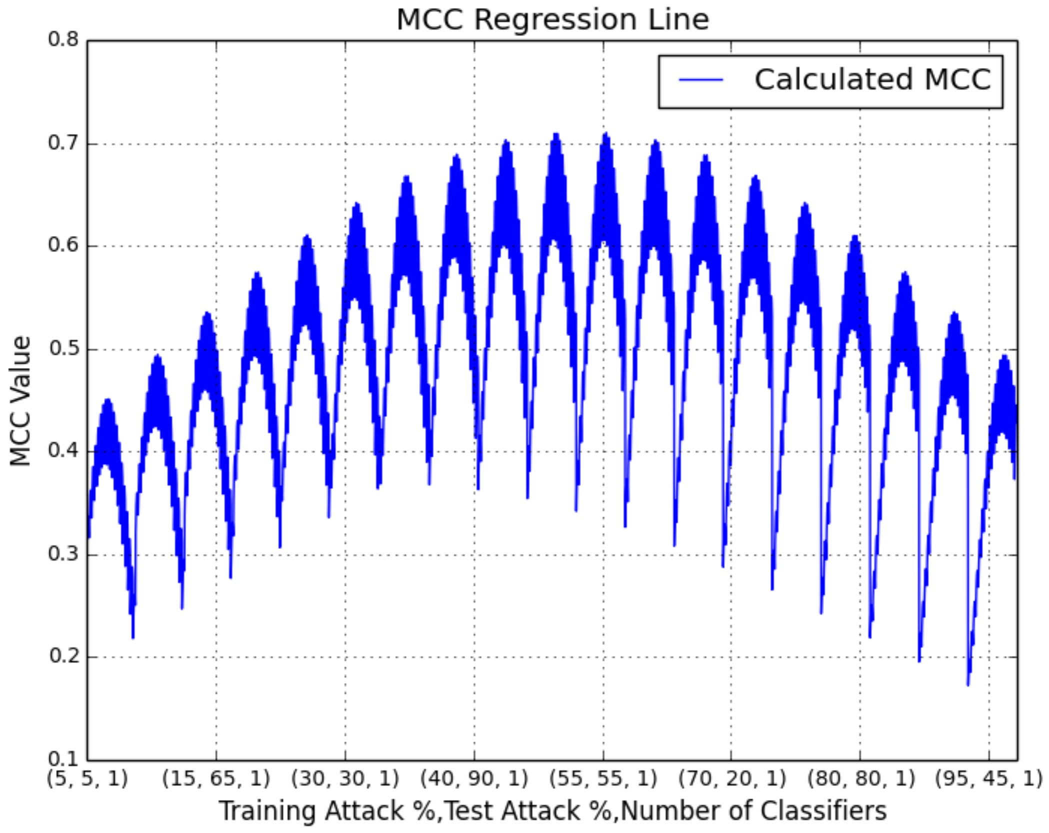


Figure 18. MCC Regression Line Plot

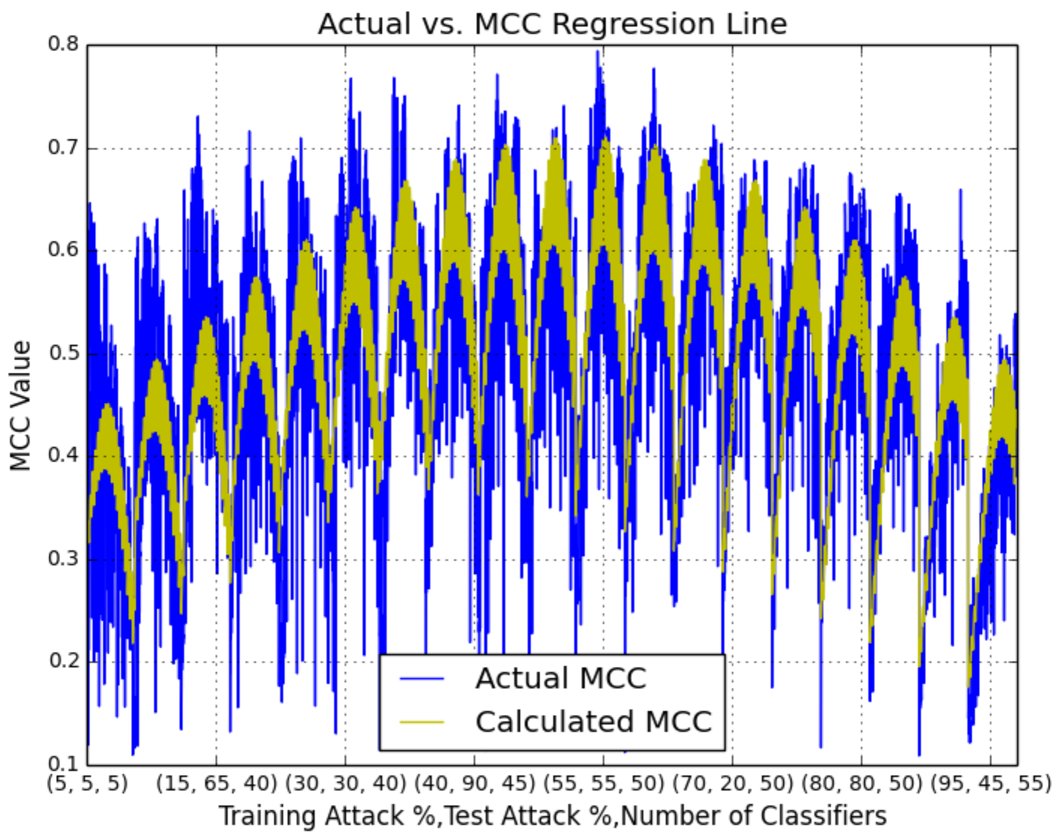


Figure 19. MCC Regression Line Plot Overlaid on Experimental Data

In Figure 18 what appear to be global maxima can be seen around the percentage of training attacks being equal to 50% and again at 55%. This may also be seen in the top ten MCC values in Table 8. Table 8 also shows that the topmost MCC values are actually quite close to each other, which explains the multiple peaks of similar value in Figure 18. But there are significant curves near these points that bear examination. A magnified view can be seen in Figure 20.

Table 8
Top Ten Sorted Calculated MCC Values for Overall Regression Line

Training Attack %	**Test Attack %**	**Number of Classifiers**	**Calculated MCC**
55	60	60	0.709912
55	60	65	0.709795
50	60	60	0.709152
50	55	60	0.709074
50	60	65	0.709034
50	55	65	0.708818
55	60	55	0.708516
55	60	70	0.708165
50	55	55	0.707818
50	60	55	0.707757

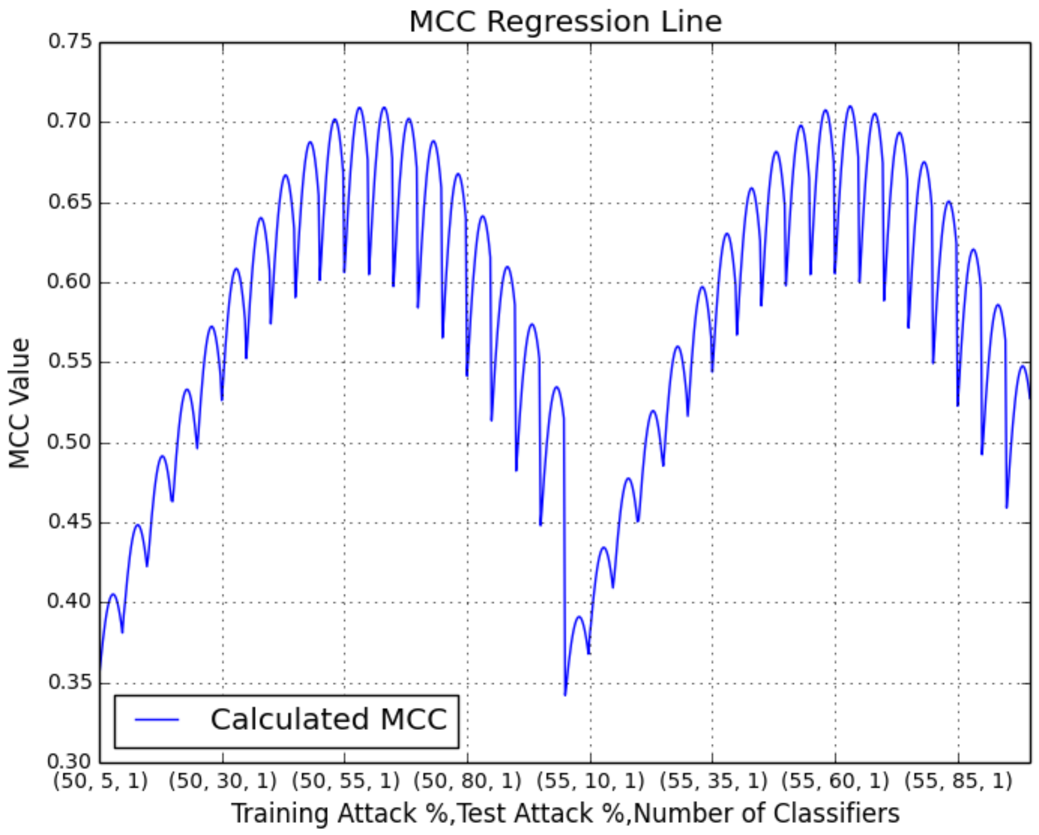


Figure 20. Magnified MCC Regression Line Plot

In the magnified view, as before, two global peak MCC values occur where the training attack percentage is 55% and again at 60%. The corresponding test attack percentages for these two points occur at 55% and 60%. This is confirmed by Table 9 which shows the top ten highest MCC values. Again, the calculated MCC values for the topmost points are very close in value. The two peaks have a different number of classifier values, being 60 and 65 classifiers. But there is still the oscillation in the curve to explain. This appears to coincide with the number of classifiers. And indeed, Figure 21 in which the training attack percentage is fixed at 50%, the test attack percentage is fixed at 55%, and the number of classifiers is allowed to vary, shows this. The influence of the varying number of classifiers can be seen to be correlated with the rise and fall of the curve. As can be seen, a peak occurs at approximately 60 classifiers.

Table 9
Top Ten Sorted Calculated MCC Values

Training Attack %	**Test Attack %**	**Number of Classifiers**	**Calculated MCC**
55	60	60	0.71100
55	60	65	0.71089
50	60	60	0.71020
50	55	60	0.71011
50	60	65	0.71010
50	55	65	0.70984
55	60	55	0.70955
55	60	70	0.70922
50	55	55	0.70883
50	60	55	0.70876

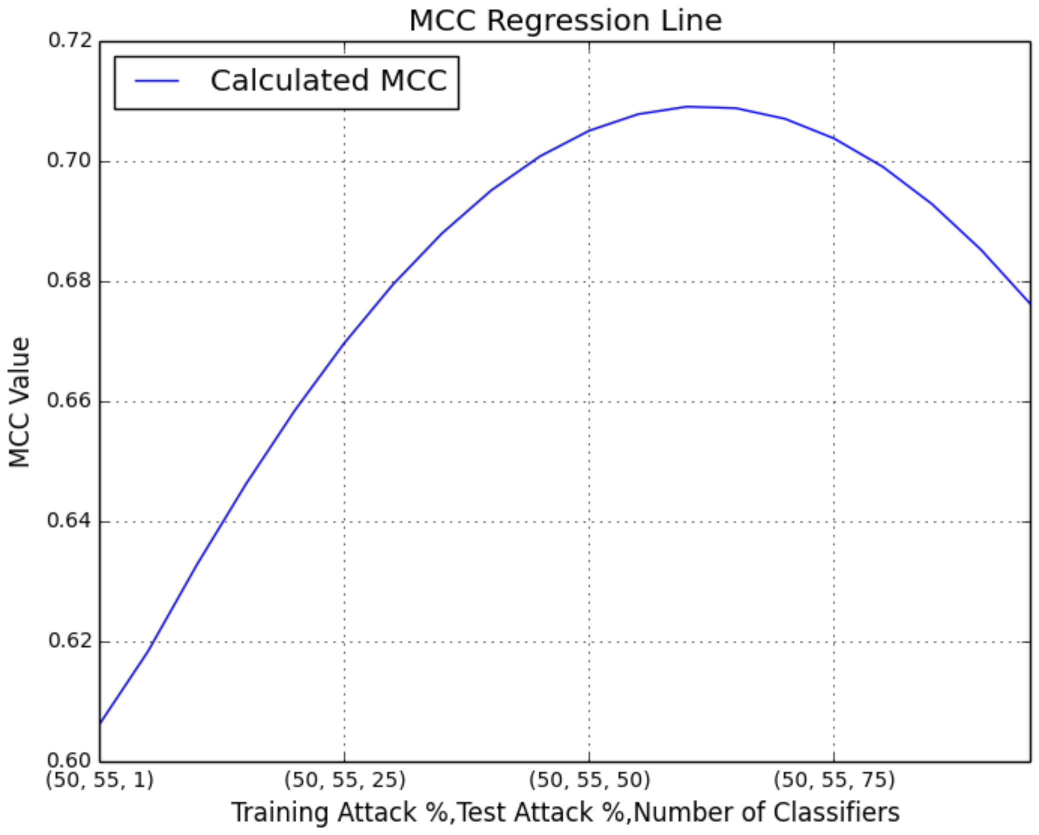


Figure 21. Increased Magnification MCC Regression Line

But would this relationship hold for an environment in which there were relatively few attacks, for example a test percentage of 5%? This is an important consideration since a classifier may be called on to operate in a wide variety of attack prevalence. The plot that results when the test percentage is held fixed at 5% and the training percentage and number of classifiers are allowed to cycle through their full range is shown in Figure 22. Table 10 shows the top ten calculated values. Both reveal peaks at 35% training attacks, 5% test attacks, and 55-60 classifier points. Although the optimal training percentage is different than other points on the curve, there is overlap between the number of classifiers for the top entries in Table 8, Table 9 and Table 10. The top 5 entries of Table 8 fall in the range of 60 to 65 classifiers. The top 5 of Table 9 also fall in the 60 to 65 classifier range. For Table 10, the top 5 fall in the 55 to 60 classifier range. As noted before, the MCC values of the top entries of these tables are very close to each other. Consequently, for this example, an ensemble of approximately 60 classifiers might work well across different ranges of training and test attack percentages. The practical value of this observation is that it suggests that a given ensemble classifier can potentially be trained for a particular attack environment by varying the number of attacks in the training phase, rather than varying the number of classifiers, at least once a semi-optimal number of classifiers has been selected.

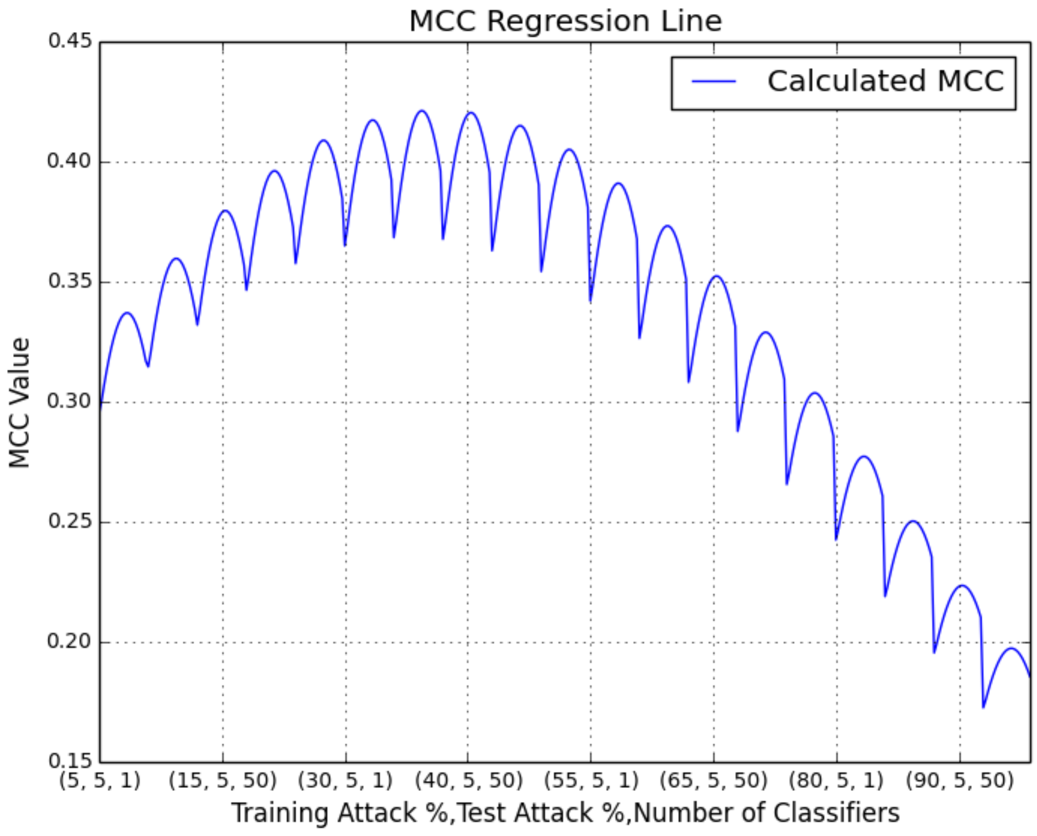


Figure 22. Magnified Alternate Point on MCC Regression Line Plot

Table 10
Top Ten Sorted Calculated MCC Values for 5% Test Attacks

Training Attack %	**Test Attack %**	**Number of Classifiers**	**Calculated MCC**
35	5	55	0.421168
35	5	60	0.421090
40	5	55	0.420382
35	5	50	0.420348
40	5	60	0.420304
35	5	65	0.420115
40	5	50	0.419563
40	5	65	0.419330
35	5	45	0.418636
35	5	70	0.418248

As an example, what if the percentage of test attacks remained at 5%, the number of classifiers was fixed at 60, and the percentage of training attacks was allowed to vary? Figure 23 answers this question. There is a peak at training attacks at 35%, test attacks at 5%, and the number of classifiers equal to 60.

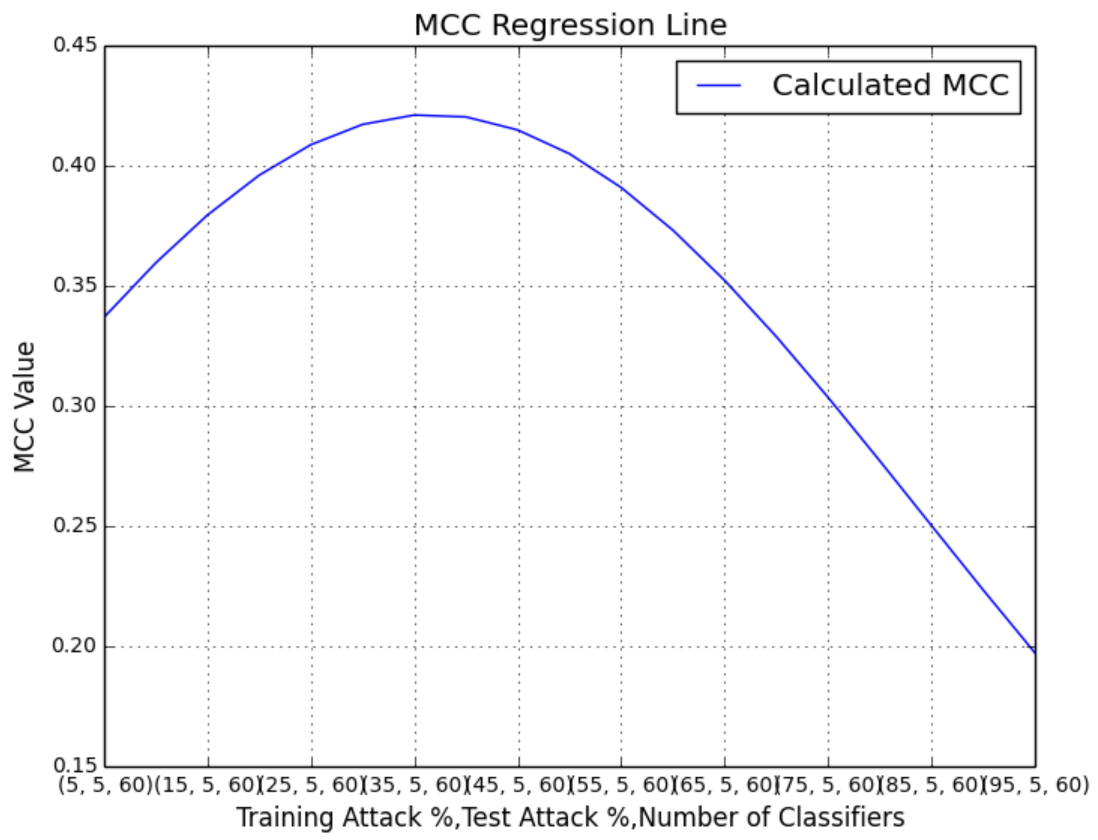


Figure 23. MCC Plot with Test Attacks = 5% and Number of Classifiers = 60

The practical result of the foregoing is that it provides a model for optimally configuring the detection ensemble. The approximately 60 classifier optimal point is consistent across the regression line. If this point is fixed, and the percentage of test attacks is set to a particular design goal, which in the example just given is 5%, then the regression equation can be used to calculate the optimal percentage of attacks in the training set. The regression equation as given previously as Equation 7 is repeated here:

$$\begin{aligned} ln(exp\_posttest\_mcc) &= -1.37911 + 0.0156883 \\ &* training\_percent\_attacks + 0.0163151 \\ &* test\_percent\_attacks + 0.00483186 \\ &* number\_classifiers - 0.000222984 \\ &* training\_percent\_attacks^2 - 0.000203297 \\ &* test\_percent\_attacks^2 - 0.0000426789 \\ &* number\_classifiers^2 + 0.000132323 \\ &* training\_percent\_attacks * test\_percent\_attacks \\ &+ 0.00000783217 * test\_percent\_attacks \\ &* number\_classifiers \end{aligned} \tag{8}$$

As an example of how the training attack percentage could be calculated, assume the following:

- $test\_percent\_attacks = 5\%$
- $number\_classifiers = 60$

Substituting these values into Equation 8 produces the following:

$$\begin{aligned} ln(exp\_posttest\_mcc) &= -1.37911 + 0.0156883 * training\_percent\_attacks \\ &+ 0.0163151 * 5 + 0.00483186 * 60 - 0.000222984 \\ &* training\_percent\_attacks^2 - 0.000203297 * 5^2 \\ &- 0.0000426789 * 60^2 + 0.000132323 \\ &* training\_percent\_attacks * 5 + 0.00000783217 * 5 \\ &* 60 \end{aligned} \tag{9}$$

A simple substitution of $f(training\_percent\_attacks) = ln(exp\_posttest\_mcc)$ gives:

$$\begin{aligned} f(&training\_percent\_attacks) \\ &= -1.37911 + 0.0156883 * training\_percent\_attacks \\ &+ 0.0163151 * 5 + 0.00483186 * 60 - 0.000222984 \\ &* training\_percent\_attacks^2 - 0.000203297 * 5^2 \\ &- 0.0000426789 * 60^2 + 0.000132323 \\ &* training\_percent\_attacks * 5 + 0.00000783217 * 5 * 60 \end{aligned} \tag{10}$$

Since the goal is to maximize the right hand side of the equation to give the highest value of MCC, calculating the first derivate of the equation, setting it to zero, and then solving for $training\_percent\_attacks$ will allow the maxima of the equation to be found (Swokowski, 1983). Calculating the first derivative of both sides utilizing yields:

$$\begin{aligned} f'(&training\_percent\_attacks) \\ &= -0.000445968 * training\_percent\_attacks + 0.016349915 \end{aligned} \tag{11}$$

In order to obtain the points at which y will be a maximum, $f'(dtraining\_percent\_attacks)$ is set equal to 0.

$$\begin{aligned} f'(&dtraining\_percent\_attacks) = \\ &= -0.000445968 * training\_percent\_attacks + 0.016349915 \\ &= 0 \end{aligned} \tag{12}$$

Solving for$training_percent_attacks$:

$$training\_percent\_attacks = -0.016349915/-0.000445968 = 36.6616326731963 \tag{13}$$

Depending on the size of the training set, this number will result in a fractional attack which cannot exist. Consequently, the product of this percentage and the number of instances in the training set will have to be either truncated or rounded to the nearest integer value. To illustrate this, each training set in this research contained 1,765 instances. Multiplying 1,765 by 36.6116324680454% yields 646.19531306099425. This would be rounded to 646 instances. The fraction of training attack instances to all instances is given by 646/1,765 and is approximately equal to 36.6%. Substituting 36.6% back into the regression equation as $training\_percent\_attacks$ and calculating the inverse log function yields an MCC value of 0.421349226707. A value of 36.6 is very close to the value of 35, which was estimated from Figure 23 and the values of Table 10 which had been calculated with a granularity of 5 attacks. Naturally, some experimentation in actual implementation would be necessary, but this method provides a starting point. It would also be possible to use this same method to instead select an arbitrary training percentage and select an optimal number of classifiers, if a fixed attack percentage had been selected. One thing that was not explored in this research was the relation of how many unique JSON attributes there were in aggregate and how that might affect the number of classifiers required for a given level of performance. This is left for future research.

Performance by Attack Type

One question that has yet to be answered is whether the approach described is equally capable of detecting all attack types, rather than performing better on some types of attacks than others. To judge how well specific attacks may be detected, during each experimental trial the type of attack present and whether or not an attack was detected for each data instance was recorded. Unfortunately, this is not enough information to measure performance using the techniques of MCC calculation or the ROC plot. Determination of performance by attack type is not straightforward. In fact, calculating the MCC is not possible, nor is creating an ROC plot. The reason for this is that it is only possible to know the number of True Positives (TP) and False Negatives (FN) for a specific attack type. If an attack of a specific type is present and it is successfully detected, then this is a True Positive (TP). If an attack of a specific type is present and it is not detected, then it is a False Negative (FN). Conversely, if an attack of a specific type is not present, and an attack is detected, then this is a False Positive (FP). But the problem is for which attack type is it a False Positive? The classifier is trained to detect eight different types of attacks. Which attack type is not present? Since the type of attack that is not present is indeterminate, it is not possible to calculate the number of False Positives (FP) for a specific type of attack. Likewise, if no attack is present and no attack is detected, which is a True Negative (TN), which type of attack was not present? Again, there are eight different types of attacks that may not have been present. Accordingly, it is not possible to calculate the number of True Negatives (TN). Therefore, the MCC cannot be calculated as defined by Equation 3, without the FP and TN values.

As for creating an ROC plot, the y-axis is the True Positive Rate (TPR). The True Positive Rate (TPR) is the ratio of correctly identified positives to all positives, given by $TPR = TP/\,(TP + FN)$ (Kelleher, MacNamee, & D'Arcy, 2015, p. 414). The total number of attacks of a specific type present is known, that is the number of correctly detected attacks plus those that were really attacks but were missed or $(TP + FN)$. Those which were detected (TP), is known. Therefore, it is possible to calculate the TPR. However, the x-axis of the ROC plot is the False Positive Rate (FPR). The FPR is given by $FPR = FP/(TN + FP)$ (Kelleher, MacNamee, & D'Arcy, 2015, p. 414). As discussed, the numbers of False Positives (FP) and True Negatives (TN) are indeterminate. Thus, construction of the typical ROC plot is not possible.

Given the preceding, it is at least possible to examine the TPR or, in this context, the proportion of actual attacks that were detected. The FNR will not be examined here since it is trivially related to the TPR by $FNR = 1 - TPR$ (Kelleher, MacNamee, & D'Arcy, 2015, p. 414). The results for the TPR are shown in a boxplot in Figure 24.

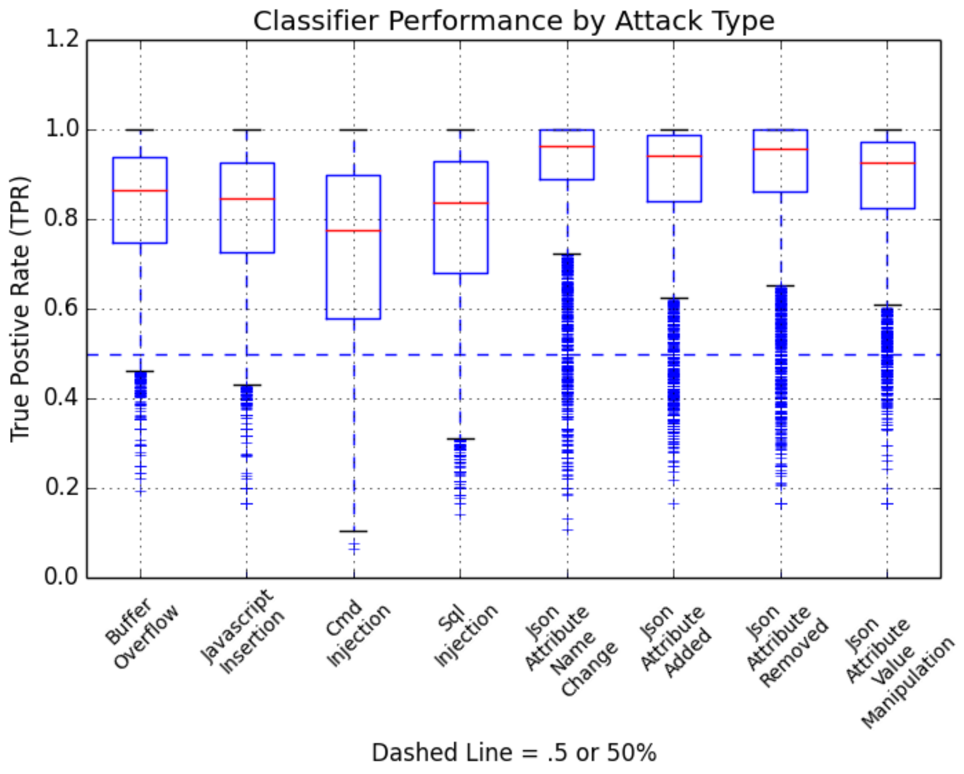


Figure 24. Classifier Performance by Attack Type

The dashed line indicates a 0.5 probability or 50% chance expectation. As can be seen, the JSON attribute name change, JSON attribute added, JSON attribute removed, and JSON attribute value manipulation attacks have whiskers indicating the cutoff point for outliers all well above the 50% chance expectation line. Detection for these types of attacks is best. The buffer overflow and JavaScript insertion attacks have whiskers that fall just below the 50% line. Since these whiskers fall below the 50% line, there are some points which are not outliers that may fall in the range of chance expectation. However, the box which indicates the middle 50% of the distribution is fully inside the 50% chance expectation line and since the whiskers are close to this line, most of the distribution will be at points better than chance expectation. The SQL injection attacks are in a similar situation, though the whiskers fall some distance to the low side of the 50% chance expectation line. And finally with command injection attacks, while the box which indicates 50% of the points is solidly above the 50% chance expectation line, more points of this distribution will fall below the 50% line. Consequently, a rough ordering from best to worst performance in terms the odds of a particular attack type being detected is:

- JSON Attribute Name Change
- JSON Attribute Removed
- JSON Attribute Added
- JSON Attribute Value Manipulation
- Buffer Overflow
- JavaScript Insertion
- SQL Injection
- Command Injection

One possible explanation of this ordering is that structural anomalies are very definite. That is, it is easy to tell if there has been any modification. The detection of other attack types may rely on suspicious n-grams, values, or other subtler indications. Command injection did stand out as the lowest-performing attack type, though there was a substantial number of example attacks of this kind used in the research. One possibility is that the injected commands may tend to be shorter and harder to recognize, though that would require further research and could potentially also be said about the other lower-performing attack types such as JavaScript insertion and SQL Injection. Another possibility is that the Command Injection attacks are similar to valid data. Either of these possibilities could increase the difficulty of detecting these attacks.

As mentioned in Chapter 3 and shown in Table 5, there were major disparities in the numbers of available templated attacks, which are attacks that are made by essentially copying a fixed string into a random point in the attacked data structure. These disparities did not seem to have much relation to attack detection rates by type. Taken from the preceding list, the ranked order of performance from best to worst of the templated attacks is:

- Buffer Overflow (number of templates: 2,659)
- JavaScript Insertion (number of templates: 78)
- SQL Injection (number of templates: 377)
- Command Injection (number of templates: 1,276)

As can be seen, there is no apparent simple linear correlation between performance rank and the number of templates that were available. Attacks with the highest numbers of available templates are both at the top and the bottom of the list. It is also interesting to rank the attacks in terms of the variance in order from most to least as seen in Figure 24:

- Command Injection (number of templates: 1,276)
- SQL Injection (number of templates: 377)
- JavaScript Insertion (number of templates: 78)
- Buffer Overflow (number of templates: 2,659)

Again, there is no apparent simple linear relation, such as the more the templates available, the smaller the variance. Instead, the attack types that have the least and the most variance find themselves at both ends of the range of variances seen.

An additional question is whether the ability to detect different attack types depends on other parameters, such as the percentage of attacks in the training and test datasets as well as the number of classifiers. It is informative to examine the proportions of certain types of attacks plotted against these three parameters, in sorted order in Figure 25.

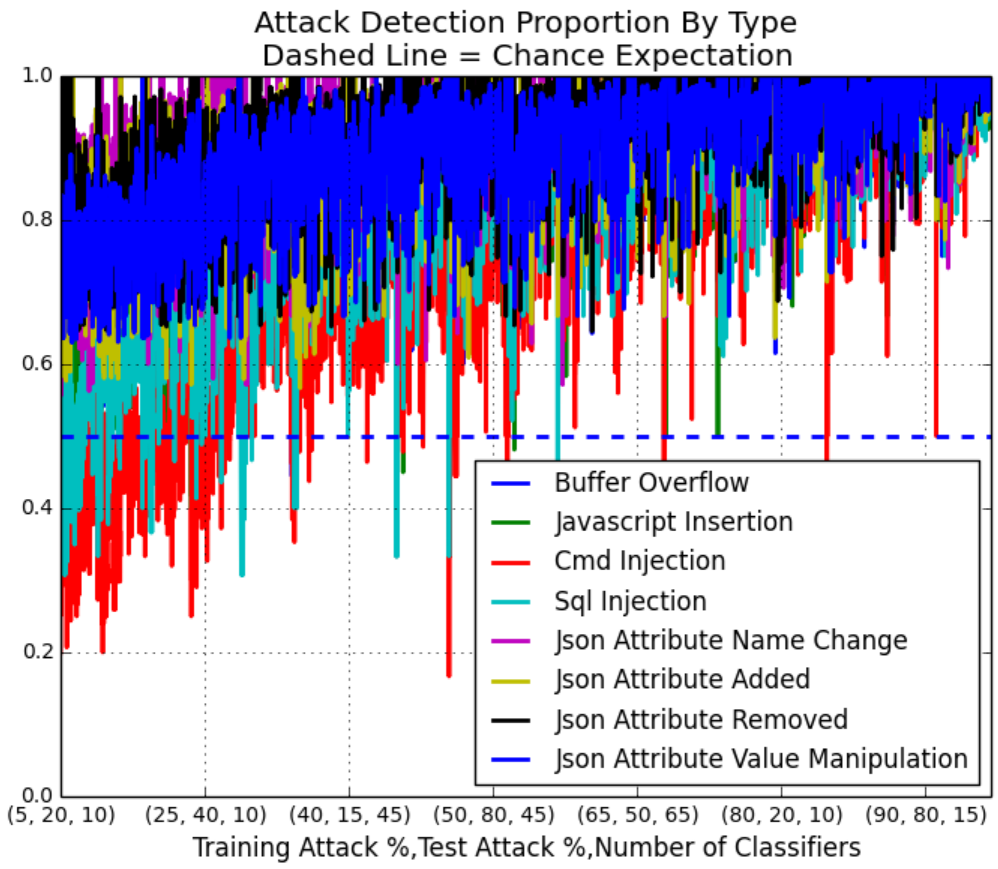


Figure 25. Attack Detection Proportion by Type

The rate of detection for each of the types follows a more or less upward trend with increasing percentages of attacks in the training set. As was noted in Figure 24, the SQL Injection and Command Injection attacks are worst performing since they are the bottommost lines in Figure 25. Excursions below the chance expectation line occur in both of these series. Of note is that there is not a maximum in the middle where there is optimal performance as there was with the MCC in Figure 18. This is because Figure 25 is showing what is effectively the True Positive Rate. It is possible to have a high True Positive Rate, but still have a low MCC value since the MCC measure takes into account False Positives (FP) and True Negatives (TN), which are not parameters used in calculating the TPR. Consequently, the MCC and TPR do not always align.

Feature Importance

A remaining question is how important each of the metrics of the Jaccard Coefficient, attack and non-attack n-gram counts, Shannon entropy, and overall JSON object length were to the detection of attacks. If it were the case that one metric alone dominated, then it might argue against the data fusion approach across several metrics. To answer this question, one of the outputs of the Scikit-learn Random Forest Classifier, namely that of feature importance was used. The Random Forest Classifier provides a list of weights for each attribute indicating its importance in the classification process. To summarize the feature importance in aggregate, the weights for each of the features were summed and sorted into a top 30 feature list. Beyond 30 features, the weights were vanishingly small. Actual values may be seen in Appendix F. A graph of these values can be seen in Figure 26. Specific feature names here have been replaced with a generic equivalent such as “Feature 0” or “Feature 1.”

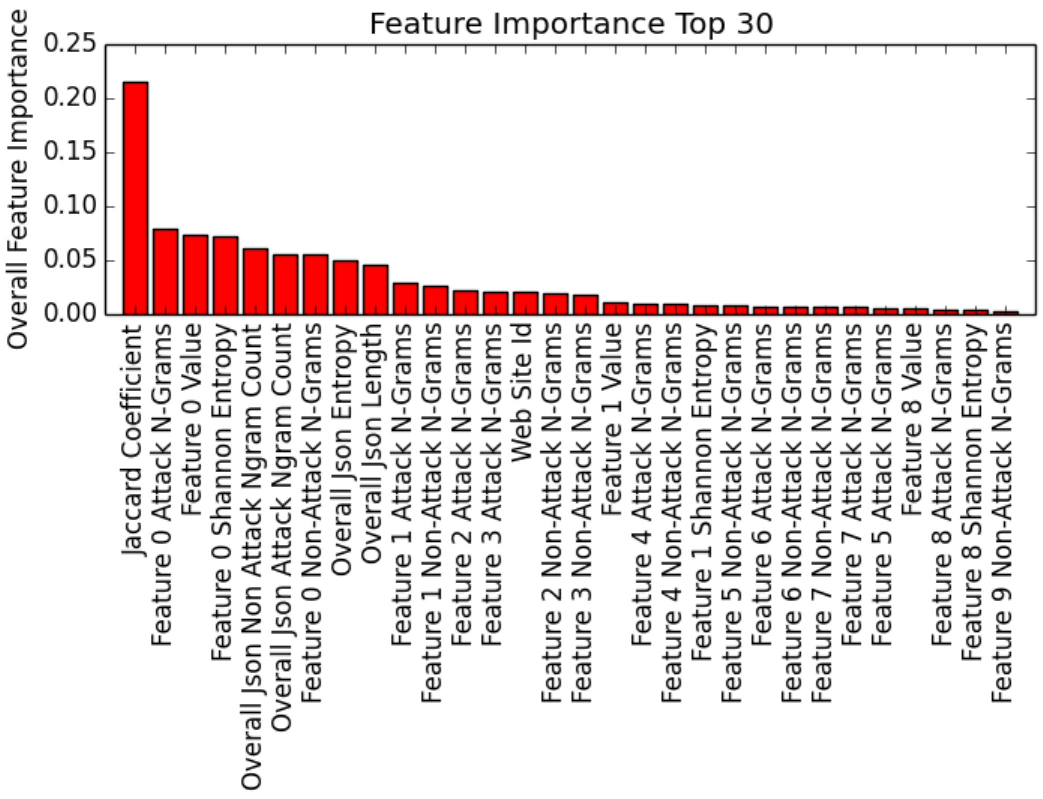


Figure 26. Feature Importance

A significant finding is that the Jaccard Coefficient is quite important in the detection of attacks. If structures are tampered with, there is an excellent chance that an attack occurred. In terms of feature importance, the Jaccard Coefficient is close to three times the importance of the next closest metric. The Jaccard Coefficient is a "whole instance" metric, rather than the next three that follow in the chart, which are specific to a particular JSON attributes, being the number of attack n-grams followed by the value of a specific attribute. But the next two most important features are again "whole instance," rather than attribute specific. These are the instance-wide non-attack and attack n-gram counts, respectively. Then there is an intervening attribute-specific metric. But then it is back to two whole instance metrics, which are the Shannon entropy and the JSON object length for the entire instance. Past this point, with the exception of the web site id in 14th place, it is all largely measures of attack and non-attack n-gram counts in specific attributes, attribute-specific Shannon entropy values, and attribute values.

One interpretation of these numbers, aside from the Jaccard Index importance, is that there is a mixture of whole instance metrics and metrics for specific JSON object attributes. One of the possibilities considered in this research was that the whole instance metrics would dominate and make attribute-specific metrics irrelevant. However, that was not the case. N-gram counts turned out to be very effective, as previous researchers reported detection (Wang et al., 2006; Wressnegger et al., 2013; Choi et al., 2009; Choi et al., 2012). Shannon entropy was found to be an important metric, but it was less important than n-gram counts. Feature values are not to be left out though, since there were three in the top 30 features. And the web site identifier did turn out to be a factor of medium importance. It was suspected that if there were variations between web sites, including the web site could help in partitioning unique metric values for that web site. Still, the web site id was ranked in the 14th position, so it was not one of the most important for detecting attacks.

Chapter 5: Conclusions

The ability of some configurations of a Random Forest ensemble classifier to successfully detect attacks in JSON formatted data as a rate better than chance expectation while using a data fusion approach of different metrics was demonstrated. Consequently, the basic goal of this research was achieved. This goal was also exceeded since the detection capability of some experimental configurations used in this research was substantially better than chance expectation. While some attacks were more successfully detected than others, all fell within a statistically significant range, and could be moved into a higher performance range by the careful selection of the percentage of attacks used in training and the number of classifiers used. Furthermore, a design model was proposed to assist in the selection of these operating parameters. In addition, the method developed here is not dependent on particular signatures, but is based in anomaly detection. As such, it has the potential to detect even newly created threats. Support was also demonstrated for the data fusion approach since the feature importance showed a mix of metrics that dealt with the entire instance, as well as specific JSON attributes.

Another important conclusion of this research is that attacks on JSON structures, rather than just attribute values, can be a solid indication of attack, as evidenced by the performance of the trained classifier for JSON structural anomalies. This was supported by the performance of the Jaccard coefficient as a reliable indicator of attack.

An additional important outcome of this research is that the approach of looking at the individual elements of JSON structures, rather than just overall metrics of the text that makes up a JSON object, has value. In examining feature importance, the Jaccard coefficient was the best-performing, but this was closely followed by attack metrics that were focused on individual parts of a JSON structure. This validates the holistic approach of considering both overall as well as fine-grained metrics from the elements of a JSON data structure.

This research suggests many different possible avenues for future efforts. Some possibilities for further research include expanding the variety of datasets used, exploring the continuous learning of attack and non-attack JSON data instances, and the evaluation of different n-gram lengths, and the use of a word level n-gram approach versus the character level n-gram approach used here. Time and memory optimization of the classification process were not addressed in this research, so future research to improve performance could include experimenting with the use of dimensionality reduction or perhaps taking the random forest classifier implementation feature ranking and then repeating the training process with the topmost important features in a two-pass approach. As noted in Chapter 4, this research did not explore the effect of the number of unique JSON parameters on the number of classifiers for a given level of performance. This could be another possible future research topic. Other possible areas of research could include experimentally modifying the parameters the Random Forest algorithm uses to build its decision trees. Yet another research direction could be comparing the ability of a classifier trained for multiple attack types to a classifier trained for a single type of attack. The present research considered an ensemble of classifiers for multiple types of attacks. An alternate approach would be an ensemble of classifiers where each classifier was trained for a specific attack type.

A further possible topic, though removed from the detection of attacks, would be to use the approach described here to look for anomalies in the log files of systems that use the

JSON format to record system activity. In this application, the detection of anomalous data would be used to detect malfunction instead of attacks.

Since JSON is becoming more and more popular as a data format, attacks on Web applications and services using it are likely to increase. Furthermore, since it is unreasonable to expect all implementations to be retroactively modified to be more resistant to attack, developing a mechanism to protect vulnerable services from attack, as has been proposed in this research, has merit. Moreover, this research has suggested many opportunities for further investigation.

References

Anderson, J., Lehnardt, J., & Slater, N. (2010). *CouchDB the Definitive Guide*. Sebastopol: O'Reilly Media, Inc. Retrieved from http://public.eblib.com/choice/publicfullrecord.aspx?p=536776

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. F., & Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, *16*(5), 412–424. doi:10.1093/bioinformatics/16.5.412

Bangia, R. (2010). *Dictionary of information technology*. New Delhi: Firewall Media.

Berners-Lee, T., Cailliau, R., Groff, J., & Pollermann, B. (2010). World-wide web: the information universe. *Internet Research*, 20(4), 461–471. doi:10.1108/10662241011059471

Berners-Lee, T., & Connolly, D. (1993, June). Hypertext Markup Language (HTML). IIIR Working Group. Retrieved from http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt

Bhattacharyya, D., & Kalita, J. (2014). *Network anomaly detection: a machine learning perspective*. Boca Raton: CRC Press, Taylor & Francis Group.

Blaikie, N. W. H. (2003). *Analyzing quantitative data: from description to explanation*. London ; Thousand Oaks, CA: Sage Publications Ltd.

Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format. Retrieved from http://tools.ietf.org/html/rfc7159.html

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008, November 26). Extensible Markup Language (XML) 1.0 (Fifth Edition). Retrieved December 1, 2014, from http://www.w3.org/TR/2008/REC-xml-20081126/

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

Choi, J.-H., Choi, C., Ko, B.-K., & Kim, P.-K. (2012). Detection of cross site scripting attack in wireless networks using n-Gram and SVM. *Mobile Information Systems*, 8(3), 275–286.

Choi, Y., Kim, T., Choi, S., & Lee, C. (2009). Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. In *Future Generation Information Technology* (pp. 160–172). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-10509-8_19

Collins, M. (2014). *Network security through data analysis building situational awareness*. Sebastopol, CA: O'Reilly Media. Retrieved from http://proquest.safaribooksonline.com/?fpi=9781449357894

Crockford, D. (2008). *The good parts: working with the shallow grain of JavaScript*. Farnham: O'Reilly.

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2), 222–232. doi:10.1109/TSE.1987.232894

Dierks, T., & Rescorla, E. (2008). The transport layer security (TLS) protocol version 1.2. Retrieved from https://tools.ietf.org/html/rfc5246

Draper, N. R., & Smith, H. (1998). *Applied regression analysis* (3rd ed). New York: Wiley.

ECMAScript Language Specification. (2011, June). Ecma International. Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Evans, J. D. (1996). *Straightforward statistics for the behavioral sciences*. Pacific Grove: Brooks/Cole Pub. Co.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol–HTTP/1.1. RFC 2616, June. Retrieved from http://tools.ietf.org/pdf/rfc2616.pdf

Firefox Developer Edition. (n.d.). (Version 44.0a2). Mozilla Foundation.

Galiegue, F., Zyp, K., & Court, G. (2013, January 31). JSON Schema: core definitions and terminology draft-zyp-json-schema-04. Retrieved from http://tools.ietf.org/pdf/draft-zyp-json-schema-04.pdf

Gormley, C., & Tong, Z. (2015). *Elasticsearch: the definitive guide ; [a distributed real-time search and analytics engine]* (1. ed). Beijing: O'Reilly Media.

Gray, J. (2005). A Conversation with Tim Bray. *Queue*, 3(1), 20–25. http://doi.org/10.1145/1046931.1046941

Gregory, P. H. (2010). *CISSP guide to security essentials.* Australia ; Boston, MA: Course Technology/Cengage Learning.

Grinstead, C. M., Snell, J. L. (1997). *Introduction to probability* (2nd rev. ed). Providence, RI: American Mathematical Society.

Gupta, B. C., & Guttman, I. (2013). *Statistics and probability with applications for engineers and scientists*. Hoboken, New Jersey: John Wiley & Sons, Inc.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Arlington, VA, USA (pp. 13–15).

Han, J., & Kamber, M. (2006). *Data mining: concepts and techniques* (2nd ed). Amsterdam ; Boston : San Francisco, CA: Elsevier ; Morgan Kaufmann.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (2nd ed). New York, NY: Springer.

Hochstein, L. (2014*). Ansible up & running: automating configuration management and deployment the easy way*. Beijing; Sebastopol, CA: O'Reilly. Retrieved from http://proquest.safaribooksonline.com/?fpi=9781491915318

Howard, M. (2009). Improving software security by eliminating the CWE top 25 Vulnerabilities. *Security & Privacy*, IEEE, 7(3), 68–71.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90–95. http://doi.org/10.1109/MCSE.2007.55

Hsu, F.-H., Huang, C.-H., Hsu, C.-H., Ou, C.-W., Chen, L.-H., & Chiu, P.-C. (2010). HSP: A solution against heap sprays. *Journal of Systems and Software*, 83(11), 2227–2236. doi:10.1016/j.jss.2010.06.045

Hsu, J. Y. (2012). *Computer logic: design principles and applications.* New York, NY: Springer.

Ingham, K. L., & Inoue, H. (2007). Comparing anomaly detection techniques for http. In *Recent Advances in Intrusion Detection* (pp. 42–62). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-74320-0_3

Jiang, S., Song, X., Wang, H., Han, J.-J., & Li, Q.-H. (2006). A clustering-based method for unsupervised intrusion detections. *Pattern Recognition Letters*, 27(7), 802–810. doi:10.1016/j.patrec.2005.11.007

Jones, E., Oliphant, T., Peterson, P., & others. (2001). SciPy: Open source scientific tools for Python. Retrieved from http://www.scipy.org/

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015*). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. Cambridge, Massachusetts: The MIT Press.

Knapp, E. (2011). *Industrial network security: securing critical infrastructure networks for Smart Grid, SCADA, and other industrial control systems*. Waltham, MA: Syngress.

Kolo, B. (2011). *Binary and multiclass classification*. [S.l.]: Weatherford Press.

Kristol, D. M., & Montulli, L. (2000). HTTP state management mechanism. Retrieved from http://tools.ietf.org/html/rfc2965

Kruegel, C., Vigna, G., & Robertson, W. (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5), 717–738. doi:10.1016/j.comnet.2005.01.009

Lampesberger, H. (2013). A Grammatical Inference Approach to Language-Based Anomaly Detection in XML. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference* on (pp. 685–693). IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6657306

Landoll, D. J. (2011). *The security risk assessment handbook a complete guide for performing security risk assessments*. Boca Raton, FL: CRC Press. Retrieved from http://dx.doi.org/10.1201/b10937

Lee, D., & Chu, W. W. (2000). Comparative analysis of six XML schema languages. *Sigmod Record*, 29(3), 76–87.

Leedy, P. D., & Ormrod, J. E. (2010). *Practical research: planning and design* (9th ed.). Upper Saddle River, NJ: Merrill.

Li, X., & Xue, Y. (2014). A survey on server-side approaches to securing web applications. *ACM Computing Surveys*, 46(4), 1–29. doi:10.1145/2541315

Liu Peng, L. L. (2005). A review of missing data treatment methods. *Int. Journal of Intel. Inf. Manag. Syst. and Tech*, 1, 3.

Lutz, M. (2013). *Learning Python* (Fifth edition). Beijing: O'Reilly.

Lyda, R., & Hamrock, J. (2007). Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, 5(2), 40–45.

Manual:What is MediaWiki? - MediaWiki. (n.d.). Retrieved September 17, 2015, from https://www.mediawiki.org/wiki/Manual:What_is_MediaWiki%3F

Masud, M., Khan, L., & Thuraisingham, B. M. (2012). *Data mining tools for malware detection*. Boca Raton, FL: CRC Press.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 51 – 56).

Metasploit Framework Edition. (n.d.). (Version v4.11.4). Rapid7 LLC.

Mitchell, H. B. (2007). *Multi-sensor data fusion: an introduction.* Berlin ; New York: Springer.

Muntner, A. (2015, September). FuzzDB. Retrieved September 17, 2015, from https://github.com/fuzzdb-project/fuzzdb

Odvarko, J., Jain, A., & Davies, A. (2012, December 14). HTTP Archive (HAR) format, Editor's Draft August 14, 2012. Retrieved September 15, 2015, from https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1), 37–63.

PyCharm Community Edition. (2015). (Version 5.03). JetBrains s.r.o.

PyPy - What is PyPy? (n.d.). Retrieved December 24, 2015, from http://pypy.org/features.html

Raggett, D., Le Hors, A., & Jacobs, I. (1998, November 24). HTML 4.0 Specification. World Wide Web Consortium.

Rescorla, E. (2000). Http over tls. Retrieved from http://tools.ietf.org/html/rfc2818.html:

Robnik-Šikonja, M. (2004). Improving random forests. In *Machine Learning: ECML 2004* (pp. 359–370). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-30115-8_34

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., … Vo, S. (2010, April). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. National Institute of Standards and Technology. Retrieved from http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

Sassaman, L., Patterson, M. L., Bratus, S., & Shubina, A. (2011). The Halting problems of network stack insecurity. *;Login:*, December.

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science* Conference (pp. 57–61). Retrieved from https://projects.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf

Severance, C. (2012). Discovering JavaScript Object Notation. *Computer*, 45(4), 6–8. doi:10.1109/MC.2012.132

Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, 30(1), 50–64.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1), 3–55.

Smith, B. (2015). *Beginning JSON*. Berkeley, CA: Apress.

Statgraphics Centurion XVI. (2012). (Version 16.1.18). Statpoint Technologies, Inc.

Swokowski, E. W. (1983). *Calculus with analytic geometry (Alternate ed)*. Boston, Mass: Prindle, Weber & Schmidt.

SymPy Development Team. (2014). SymPy: Python library for symbolic mathematics. Retrieved from http://www.sympy.org

Tan, P.-N., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining* (1st ed.). Boston: Pearson Addison Wesley.

Terms of Use - Wikimedia Foundation. (n.d.). Retrieved September 17, 2015, from https://wikimediafoundation.org/wiki/Terms_of_Use#7._Licensing_of_Content

Trost, R. (2010). *Practical intrusion analysis: prevention and detection for the twenty-first century*. Upper Saddle River, NJ: Addison-Wesley.

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2), 22–30. http://doi.org/10.1109/MCSE.2011.37

Wang, K., Parekh, J. J., & Stolfo, S. J. (2006). Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection* (pp. 226–248). Springer. Retrieved from http://link.springer.com/chapter/10.1007/11856214_12

Watson, D. (2007). Web application attacks. *Network Security*, 2007(10), 10–14.

Windley, P. J. (2012). *The live web building event-based connections in the cloud*. Boston, Mass.: Course Technology PTR. Retrieved from http://www.books24x7.com/marc.asp?bookid=43518

Wressnegger, C., Schwenk, G., Arp, D., & Rieck, K. (2013). A close look on n -grams in intrusion detection: anomaly detection vs. classification (pp. 67–76). *ACM Press*. doi:10.1145/2517312.2517316

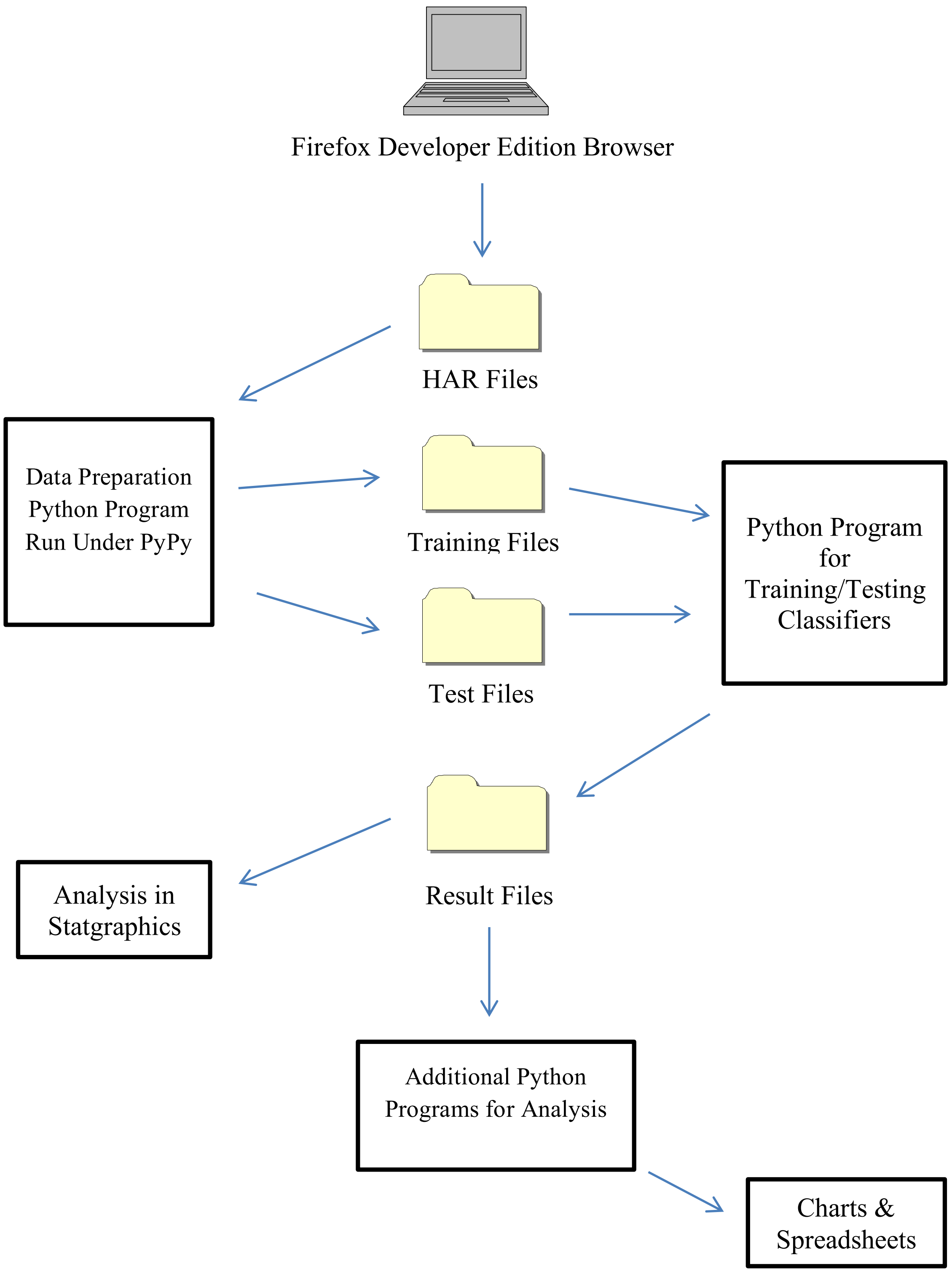
Zenil, H. (Ed.). (2011). *Randomness through computation: some answers, more questions*. Singapore: World Scientific.

APPENDICES

Appendix A: HAR File Excerpt

```
"content": {
  "mimeType": "application/json; charset=utf-8",
  "size": 769,
  "text":
  "{\"batchcomplete\":\"\",\"query\":{\"repos\":[{\"name\":\"shared\",\"displayname\":\"
  Commons\",\"rootUrl\":\"//upload.wikimedia.org/wikipedia/commons\",\"url\":\"//upl
  oad.wikimedia.org/wikipedia/commons\",\"thumbUrl\":\"//upload.wikimedia.org/wiki
  pedia/commons/thumb\",\"initialCapital\":\"\",\"descBaseUrl\":\"https://commons.wik
  imedia.org/wiki/File:\",\"scriptDirUrl\":\"https://commons.wikimedia.org/w\",\"fetch
  Description\":\"\",\"favicon\":\"/static/favicon/commons.ico\",\"canUpload\":\"\"},{\"n
  ame\":\"local\",\"displayname\":\"Wikipedia\",\"rootUrl\":\"//upload.wikimedia.org/wi
  kipedia/en\",\"local\":\"\",\"url\":\"//upload.wikimedia.org/wikipedia/en\",\"thumbUrl\
  ":\"//upload.wikimedia.org/wikipedia/en/thumb\",\"initialCapital\":\"\",\"scriptDirUrl\"
  :\"/w\",\"favicon\":\"https://en.wikipedia.org/static/favicon/wikipedia.ico\",\"canUploa
  d\":\"\"}]}}"
},
```

Appendix B: Experimental Flow


Firefox Developer Edition Browser
HAR Files
Data Preparation Python Program Run Under PyPy
Training Files
Test Files
Python Program for Training/Testing Classifiers
Result Files
Analysis in Statgraphics
Additional Python Programs for Analysis
Charts & Spreadsheets

Appendix C: Statgraphics Output for N-Gram Versus Time

Multiple Regression - time_in_ms (abs(SRESIDUALS)<2.9)

Dependent variable: time_in_ms
Independent variables:
    number_characters
    ngram_length
Selection variable: abs(SRESIDUALS)<2.9

		Standard	*T*	
Parameter	*Estimate*	*Error*	*Statistic*	*P-Value*
CONSTANT	-1.50712	0.120197	-12.5387	0.0000
number_characters	0.00128497	0.0000752771	17.0698	0.0000
ngram_length	0.141355	0.00817051	17.3006	0.0000

Analysis of Variance

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
Model	189.884	2	94.9418	293.02	0.0000
Residual	49.5739	153	0.324012		
Total (Corr.)	239.457	155			

R-squared = 79.2974 percent
R-squared (adjusted for d.f.) = 79.0268 percent
Standard Error of Est. = 0.569221
Mean absolute error = 0.461623
Durbin-Watson statistic = 1.97685 (P=0.4428)
Lag 1 residual autocorrelation = 0.00356483

Stepwise regression
Method: backward selection
P-to-enter: 0.05
P-to-remove: 0.05

Step 0:
2 variables in the model. 153 d.f. for error.
R-squared = 79.30%   Adjusted R-squared = 79.03%   MSE = 0.324012

Final model selected.

The StatAdvisor
The output shows the results of fitting a multiple linear regression model to describe the relationship between time_in_ms and 2 independent variables. The equation of the fitted model is

time_in_ms = -1.50712 + 0.00128497*number_characters + 0.141355*ngram_length

Since the P-value in the ANOVA table is less than 0.05, there is a statistically significant relationship between the variables at the 95.0% confidence level.

The R-Squared statistic indicates that the model as fitted explains 79.2974% of the variability in time_in_ms. The adjusted R-squared statistic, which is more suitable for comparing models with different numbers of independent variables, is 79.0268%. The standard error of the estimate shows the standard deviation of the residuals to be 0.569221. This value can be used to construct prediction limits for new observations by selecting the Reports option from the text menu. The mean absolute error (MAE) of 0.461623 is the average value of the residuals. The Durbin-Watson (DW) statistic tests the residuals to determine if there is any significant correlation based on the order in which they occur in your data file. Since the P-value is greater than 0.05, there is no indication of serial autocorrelation in the residuals at the 95.0% confidence level.

In determining whether the model can be simplified, notice that the highest P-value on the independent variables is 0.0000, belonging to number_characters. Since the P-value is less than 0.05, that term is statistically significant at the 95.0% confidence level. Consequently, you probably don't want to remove any variables from the model.

Appendix D: Statgraphics Output for Classifier Regression Equation

Multiple Regression - log(exp_posttest_mcc) (abs(SRESIDUALS)<3)

Dependent variable: log(exp_posttest_mcc)
Independent variables:
    training_percent_attacks
    test_percent_attacks
    number_classifiers
    training_percent_attacks^2
    test_percent_attacks^2
    number_classifiers^2
    training_percent_attacks*test_percent_attacks
    test_percent_attacks*number_classifiers
    number_classifiers*training_percent_attacks
Selection variable: abs(SRESIDUALS)<3

		Standard	*T*
Parameter	*Estimate*	*Error*	*Statistic*
CONSTANT	-1.37911	0.0148524	-92.8542
training_percent_attacks	0.0156883	0.000338779	46.3083
test_percent_attacks	0.0163151	0.000365473	44.6409
number_classifiers	0.00483186	0.000301433	16.0296
training_percent_attacks^2	-0.000222984	0.00000297083	-75.058
test_percent_attacks^2	-0.000203297	0.00000300624	-67.625
number_classifiers^2	-0.0000426789	0.00000272054	-15.6877
training_percent_attacks*test_percent_attacks	0.000132323	0.00000269914	49.0242
test_percent_attacks*number_classifiers	0.00000783217	0.00000255673	3.06336

P-Value
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0022

Analysis of Variance

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
Model	258.471	8	32.3089	1838.57	0.0000
Residual	80.7294	4594	0.0175728		
Total (Corr.)	339.2	4602			

R-squared = 76.2001 percent
R-squared (adjusted for d.f.) = 76.1586 percent
Standard Error of Est. = 0.132562
Mean absolute error = 0.106044
Durbin-Watson statistic = 1.98142 (P=0.2642)
Lag 1 residual autocorrelation = 0.0092257

Stepwise regression
Method: forward selection
P-to-enter: 0.05
P-to-remove: 0.05

Step 0:

```
0 variables in the model.  4602 d.f. for error.
R-squared =  0.00%     Adjusted R-squared =   0.00%     MSE = 0.0737072

Step 1:
Adding variable training_percent_attacks*test_percent_attacks with P-to-enter =0
1 variables in the model.  4601 d.f. for error.
R-squared =  9.03%     Adjusted R-squared =   9.01%     MSE = 0.0670628

Step 2:
Adding variable training_percent_attacks^2 with P-to-enter =0
2 variables in the model.  4600 d.f. for error.
R-squared = 28.50%     Adjusted R-squared =  28.47%     MSE = 0.052723

Step 3:
Adding variable test_percent_attacks^2 with P-to-enter =0
3 variables in the model.  4599 d.f. for error.
R-squared = 54.11%     Adjusted R-squared =  54.08%     MSE = 0.0338484

Step 4:
Adding variable test_percent_attacks with P-to-enter =0
4 variables in the model.  4598 d.f. for error.
R-squared = 62.51%     Adjusted R-squared =  62.48%     MSE = 0.0276563

Step 5:
Adding variable training_percent_attacks with P-to-enter =0
5 variables in the model.  4597 d.f. for error.
R-squared = 73.50%     Adjusted R-squared =  73.47%     MSE = 0.0195514

Step 6:
Adding variable number_classifiers with P-to-enter =0
6 variables in the model.  4596 d.f. for error.
R-squared = 74.87%     Adjusted R-squared =  74.84%     MSE = 0.0185436

Step 7:
Adding variable number_classifiers^2 with P-to-enter =0
7 variables in the model.  4595 d.f. for error.
R-squared = 76.15%     Adjusted R-squared =  76.12%     MSE = 0.0176048

Step 8:
Adding variable test_percent_attacks*number_classifiers with P-to-enter =0.00218881
8 variables in the model.  4594 d.f. for error.
R-squared = 76.20%     Adjusted R-squared =  76.16%     MSE = 0.0175728

Final model selected.
```

The StatAdvisor
The output shows the results of fitting a multiple linear regression model to describe the relationship between log(exp_posttest_mcc) and 9 independent variables. The equation of the fitted model is

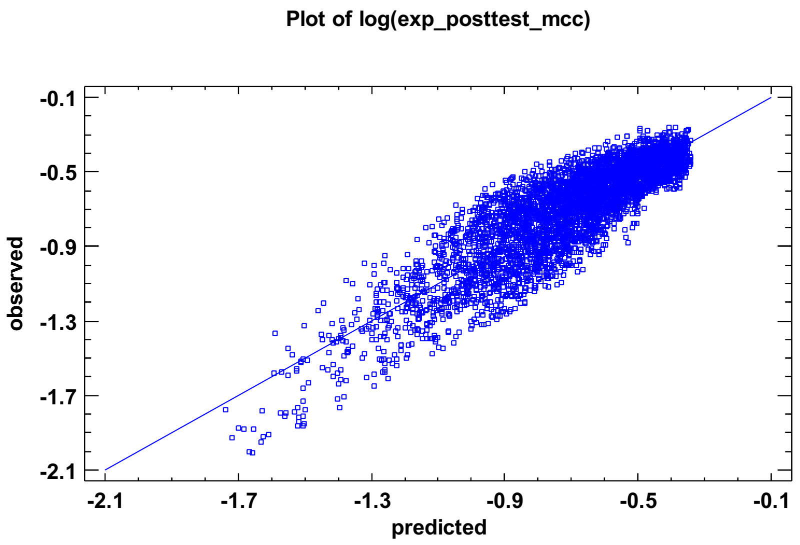
log(exp_posttest_mcc) = -1.37911 + 0.0156883*training_percent_attacks + 0.0163151*test_percent_attacks + 0.00483186*number_classifiers - 0.000222984*training_percent_attacks^2 - 0.000203297*test_percent_attacks^2 - 0.0000426789*number_classifiers^2 + 0.000132323*training_percent_attacks*test_percent_attacks + 0.00000783217*test_percent_attacks*number_classifiers

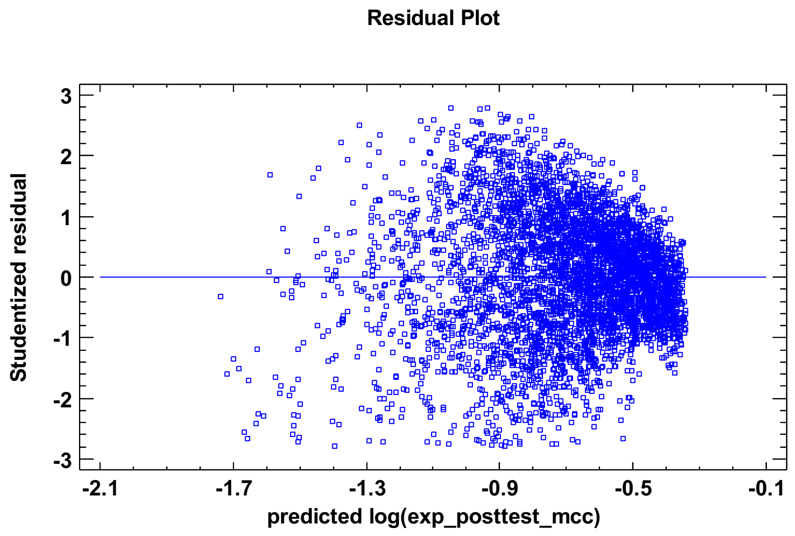
Since the P-value in the ANOVA table is less than 0.05, there is a statistically significant relationship between the variables at the 95.0% confidence level.

The R-Squared statistic indicates that the model as fitted explains 76.2001% of the variability in log(exp_posttest_mcc). The adjusted R-squared statistic, which is more suitable for comparing models with different numbers of independent variables, is 76.1586%. The standard error of the estimate shows the standard deviation of the residuals to be 0.132562. This value can be used to construct prediction limits for new observations by selecting the Reports option from the text menu. The mean absolute error (MAE) of 0.106044 is the average value of the residuals. The Durbin-Watson (DW) statistic tests the residuals to determine if there is any significant correlation based on the order in which they occur in your data file. Since the P-value is greater than 0.05, there is no indication of serial autocorrelation in the residuals at the 95.0% confidence level.

In determining whether the model can be simplified, notice that the highest P-value on the independent variables is 0.0022, belonging to test_percent_attacks*number_classifiers. Since the P-value is less than 0.05, that term is statistically significant at the 95.0% confidence level. Consequently, you probably don't want to remove any variables from the model.

Appendix E: Predicted and Residual Plots for Classifier Regression Equation





Appendix F: Feature Contribution

Feature Name	Summed Weights	Percent Contribution
jaccard_coefficient	725.094735	0.214900
_0__number_attack_ngrams_present	265.1878546	0.078595
_0__value	246.5439897	0.073070
_0__shannon_entropy	242.0645049	0.071742
raw_json_non_attack_ngram_count	204.5790808	0.060632
raw_json_attack_ngram_count	189.4452515	0.056147
_0__number_non_attack_ngrams_present	188.7165275	0.055931
raw_json_entropy	166.9860553	0.049491
raw_json_length	154.0121263	0.045645
query_pages___title_number_attack_ngrams_present	99.07064484	0.029362
query_pages___title_number_non_attack_ngrams_present	90.49726589	0.026821
query_pages___imageinfo_0__timestamp_number_attack_ngrams_present	75.98669096	0.022521
query_pages___imageinfo_0__extmetadata_DateTime_value_number_attack_ngrams_present	71.70831876	0.021253
web_site_id	69.13479002	0.020490
query_pages___imageinfo_0__timestamp_number_non_attack_ngrams_present	64.97236934	0.019256
query_pages___imageinfo_0__extmetadata_DateTime_value_number_non_attack_ngrams_present	61.42970197	0.018206
query_pages___title_value	39.03498143	0.011569
query_pages___imageinfo_0__url_number_attack_ngrams_present	33.95086289	0.010062
query_pages___imageinfo_0__url_number_non_attack_ngrams_present	33.05013924	0.009795
query_pages___title_shannon_entropy	26.07122331	0.007727
query_normalized_0__from_number_non_attack_ngrams_present	26.02634095	0.007714
query_pages___imageinfo_0__descriptionurl_number_attack_ngrams_present	25.46639811	0.007548
query_pages___imageinfo_0__descriptionurl_number_non_attack_ngrams_present	25.04200252	0.007422
query_normalized_0__to_number_non_attack_ngrams_present	24.99273121	0.007407
query_normalized_0__to_number_attack_ngrams_present	21.67484645	0.006424
query_normalized_0__from_number_attack_ngrams_present	20.83706525	0.006176

batchcomplete_value	17.49693097	0.005186
batchcomplete_number_attack_ngrams_present	12.15478325	0.003602
batchcomplete_shannon_entropy	11.94328481	0.003540
query_pages___imageinfo_0__extmetadata_ObjectName_value_number_non_attack_ngrams_present	9.88462614	0.002930