STATISTIC WHITELISTING FOR ENTERPRISE NETWORK INCIDENT RESPONSE

THESIS

Nathan E. Grunzweig, CPT, USA

AFIT-ENG-MS-16-M-019

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A:

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Army, the United States Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-16-M-019

STATISTIC WHITELISTING FOR ENTERPRISE NETWORK INCIDENT RESPONSE

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Nathan E. Grunzweig, B.S.I.T.

CPT, USA

March 2016

DISTRIBUTION STATEMENT A:

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-16-M-019

STATISTIC WHITELISTING FOR ENTERPRISE NETWORK INCIDENT RESPONSE

THESIS

Nathan E. Grunzweig, B.S.I.T.
CPT, USA

Committee Membership:

Dr. G. L. Peterson
Chair

Dr. T. H. Lacey
Member

Dr. M. J. Mendenhall
Member

AFIT-ENG-MS-16-M-019

Abstract

This research seeks to satisfy the need for the rapid evaluation of enterprise network hosts in order to identify items of significance through the introduction of a statistic whitelist based on the behavior of the processes on each host. By taking advantage of the repetition of processes and the resources they access, a whitelist can be generated using large quantities of host machines. For each process, the Modules and the TCP & UDP Connections are compared to identify which resources are most commonly accessed by each process. Results show 47% of processes receiving a whitelist score of 75% or greater in the five hosts identified as having the worst overall scores and 60% of processes when the hosts more closely match the hosts used to build the whitelist.

Acknowledgements

I would like to extend my sincerest thanks to my advisor, Dr. Gilbert Peterson, for his mentorship and unending patience. This would not have been possible without his steady guidance and support. I would also like to thank my committee for their flexibility and expertise. Lastly, I would like to thank the rest of the AFIT staff and faculty for the incredible education and opportunities afforded to me during my attendance.

Table of Contents

Page

Abstract . . . iv

Acknowledgements . . . v

List of Figures . . . viii

List of Tables . . . ix

List of Abbreviations . . . x

I. Introduction . . . 1

1.1 Background . . . 1
1.2 Motivation . . . 2
1.3 Research Goals and Hypothesis . . . 2
1.4 Approach . . . 3
1.5 Assumptions . . . 3
1.6 Contributions . . . 4
1.7 Thesis Overview . . . 4

II. Background and Related Research . . . 5

2.1 Digital Forensics . . . 5
2.2 Forensic Analysis Technologies . . . 7
2.2.1 Blacklist . . . 7
2.2.2 Whitelist . . . 7
2.2.3 Hashes . . . 8
2.3 Related Research . . . 8
2.4 Summary . . . 9

III. Methodology . . . 10

3.1 Whitelisting Overview . . . 10
3.2 Helix Data . . . 11
3.3 Pre-Processing . . . 14
3.4 Building the Whitelist . . . 16
3.5 Testing Against the Whitelist . . . 19
3.6 Generating Results . . . 21
3.7 Summary . . . 21

Page

IV. Results and Analysis . . . 23

4.1 Results . . . 23
4.1.1 #1: 131.10.16.12 . . . 24
4.1.2 #2: 131.10.16.11 . . . 25
4.1.3 #3 - #5: 131.10.157.11, 131.10.157.12, and 131.10.157.14 . . . 26
4.1.4 Further Analysis . . . 27
4.1.5 Host: 131.10.159.106 . . . 28
4.1.6 Host: 131.10.160.189 . . . 29
4.1.7 Host: 131.10.159.140 . . . 29
4.1.8 Host: 131.10.160.102 . . . 30
4.1.9 Host: 131.10.160.12 . . . 30
4.1.10 Analysis Conclusions . . . 31
4.2 Limitations . . . 31
4.3 Future Work . . . 32
4.4 Summary . . . 33

V. Conclusions and Recommendations . . . 34

5.1 Research Conclusion . . . 34
5.2 Research Contributions . . . 34

Appendix A. Code: Pre-Processing . . . 35

Appendix B. Code: Whitelist . . . 54

Appendix C. Code: Results Generation . . . 81

Appendix D. Results Extraction Scripts . . . 92

Bibliography . . . 93

List of Figures

Figure Page

1 Common Process Model of Digital Forensics [1]. 6

2 Helix Data Structure. 12

3 Helix Raw Process Data. 13

4 Aggregation Pipeline Example. 14

5 PreProcessed Data Structure. 16

6 Build Logic Flow Diagram. 18

7 Test Logic Flow Diagram. 20

8 Results Logic Flow Diagram. 22

List of Tables

Table — Page

1 Hosts with the Lowest Score. . . . 24
2 Host #1: 131.10.16.12 Process Scores. . . . 25
3 Host #2: 131.10.16.11 Process Scores. . . . 26
4 Host #3: 131.10.157.12 Process Scores. . . . 26
5 Host #4: 131.10.157.11 Process Scores. . . . 27
6 Host #5: 131.10.157.14 Process Scores. . . . 27
7 Additional Hosts Selected for Examination. . . . 28
8 Host: 131.10.159.106 Process Scores. . . . 28
9 Host: 131.10.160.189 Process Scores. . . . 29
10 Host: 131.10.159.140 Process Scores. . . . 29
11 Host: 131.10.160.102 Process Scores. . . . 30
12 Host: 131.10.160.12 Process Scores. . . . 30

List of Abbreviations

Abbreviation		Page
DoD	Department of Defense	1
DC3	DoD Cyber Crime Center	1
IT	Information Technology	1
FBI	Federal Bureau of Investigation's	1
RFCL	Regional Computer Forensics Laboratory	1
OS	Operating System	2
IT	Information Technology	5
SPH	Similarity Preserving Hashing	9
OS	Operating System	11
SQL	Structured Query Language	12
TCP	Transmission Control Protocol	12
UDP	User Datagram Protocol	12
PID	Process Identifier	12
PPID	Parent Process Identifier	15
AFIT	Air Force Institute of Technology	19
CSV	Comma-Separated Values	21
ICS	Industrial Control Systems	24
GUI	Graphical User Interface	25

STATISTIC WHITELISTING FOR ENTERPRISE NETWORK INCIDENT RESPONSE

I. Introduction

1.1 Background

THE field of Digital Forensics has steadily grown with the proliferation of computing technology. Since the early digital crimes in the 1970's the need for experienced digital forensic analysts has only grown [2]. The Department of Defense (DoD) has understood the necessity of the field of Digital Forensics with the establishment of organizations, such as the DoD Cyber Crime Center (DC3) [3]. The establishment of a strong digital forensic capability is of special importance to the DoD due to the nature of both criminal and adversarial threats.

The rapid advancement of technology has made the process of conducting digital forensic investigations increasingly complex. As enterprise Information Technology (IT) capabilities have become more inter-connected, the quantity of data that must be analyzed with each incident has grown. In 2012, the Federal Bureau of Investigation's (FBI) Regional Computer Forensics Laboratory (RFCL) reported a 40% increase in the amount of data to be analyzed in investigations [4]. However, the increase in the size and complexity of each incident has not been met with the same increase in staff and capabilities. The increase in the average quantity of forensic data associated with each case has necessitated the development of methods to quickly reduce the data that must be processed and allow the investigator to focus on items of interest.

This research addresses the need for capabilities to address the volume of data through the implementation of a whitelist based on process behaviors. In order to reduce the time required for each investigation, analysts must focus on the objects of interest that are different from the data produced by normal use and operation of devices. The development of a statistic-based whitelist highlights data of interest to investigators because of its uniqueness.

1.2 Motivation

The need for expanded Digital Forensic capabilities has grown with the proliferation of digital devices and storage. Where early forensic efforts were faced with Megabytes of data to process, modern investigations can easily amount to Terabytes of data [5]. The increase in the quantity of data collected during an incident necessitates methods of reducing the data that must be evaluated by the investigator. By developing a method of generating a statistic-based whitelist using large sets of forensic data, an analyst can better focus on the information of importance to the investigation.

1.3 Research Goals and Hypothesis

The goal of this research is to determine the viability of a statistic whitelist in reducing the amount of data that must be analyzed as part of an enterprise network incident response. There are many methods used to reduce this data but statistic-based whitelisting using large data sets has the potential to reduce the time necessary to analyze each host by quickly identifying the hosts that are most likely to yield items of interest to the investigation.

The hypothesis is based on the repetition of similar processes across hosts. This is based on how processes are executed and how the Operating System (OS) is initialized

at boot up. The behavior of the OS processes on a single host will match the OS processes of another host, provided the OS on both machines is the same. Because of this repetition of processes across many hosts, the processes that are different from the other hosts are more likely to be of forensic interest.

It is believed that this repetition of processes can be used to generate a statistic whitelist from large sets of untrusted data. By using large sets of host machines, malicious processes will represent a statistical minority within the data set. When new hosts are tested against the whitelist, any process outliers will become quickly apparent for additional analysis.

1.4 Approach

Traditional whitelists require that the data being used is of a "known good" configuration to prevent the whitelist from ignoring processes that are malicious. This research utilizes a large data set of many hosts of unknown configuration to generate a statistic-based whitelist. The intention is to take advantage of the natural repetition of processes across large quantities of hosts to mitigate the impact of malicious processes added to the whitelist.

1.5 Assumptions

This research assumes that processes behave consistently similar when the Application, its Parent process, and the application's location in the host file system all match. Additionally, it is assumed that the whitelist will be resistant to falsely ignoring malicious processes unless a significant majority of the data set is infected. It is believed that, in order for malicious processes to be falsely ignored by the whitelist, greater than half of the hosts used to build the whitelist must be infected.

1.6 Contributions

This thesis contributes to the future of forensic analysis within the DoD community. Specific contributions include the ability to process large sets of forensic data and reduce the quantity of processes that must be evaluated by a forensic analyst. As the DoD maintains both large and small networks, the ability to effectively reduce the analysis necessary to detect malicious software is invaluable.

1.7 Thesis Overview

This thesis is organized into five chapters. Chapter 2 offers a detailed perspective of forensic analysis and the methods of processing forensic data. Chapter 3 provides the methodology used which includes the configuration and tools to generate the whitelist. Chapter 4 gives a summary of results and analysis of the effectiveness of the statistic whitelist. Chapter 5 concludes this study and suggest recommendation for future work.

II. Background and Related Research

This chapter describes digital forensics and the different methods of reducing the workload of forensic analysts. Because of the incredible growth in the size of enterprise Information Technology (IT) systems, new methods of reducing forensic data must be implemented. By developing a statistic-based whitelist on the repetition of process behavior, potential items of interest to an investigation can be found by identifying processes that are different from those previously identified in the whitelist.

To further develop the justification for this research, the following sections will provide background to the previous research in this area. Section 2.1 defines digital forensics and contains an overview of the forensic process. Section 2.2 covers some of the technologies used to reduce analysis workload, including Hashes, blacklists, and Whitelists. Lastly, Section 2.3 concludes the chapter by providing details on related research.

2.1 Digital Forensics

Enterprise IT Systems have been increasingly targeted by hackers in recent years. Defending the IT infrastructure has become a priority in the military as evidenced by the recent establishment of the Cyber branch of the US Army [6]. Digital forensics capabilities are a key aspect in defense for both military and private sector networks. The field of digital forensics is essential to the detection of malicious activity within the IT infrastructure [3]. This is primarily through preemptive scanning and analysis of the network and attached devices to detect suspicious activity. Once unauthorized activity has been detected, the Incident Response process occurs.

The Common Process Model [1] by Freiling is shown in Figure 1. In this model, there are three main phases: Pre-Analysis, Analysis, and Post-Analysis. The Pre-

Analysis phase consists of all actions taken prior to analysis. The primary focus during this phase is the detection of the Incident(s) and the initial response taken. This is also when analysis of routine log data is analyzed to detect malicious activity that was not detected immediately upon infection. The Analysis phase covers all forensic data analysis of the incident and the response taken. This phase includes the collection of data from "live" machines that are still running. Lastly, the Post-Analysis phase is when all Investigative reports are completed, documenting all steps taken in the detection, capture, and analysis of the incident.

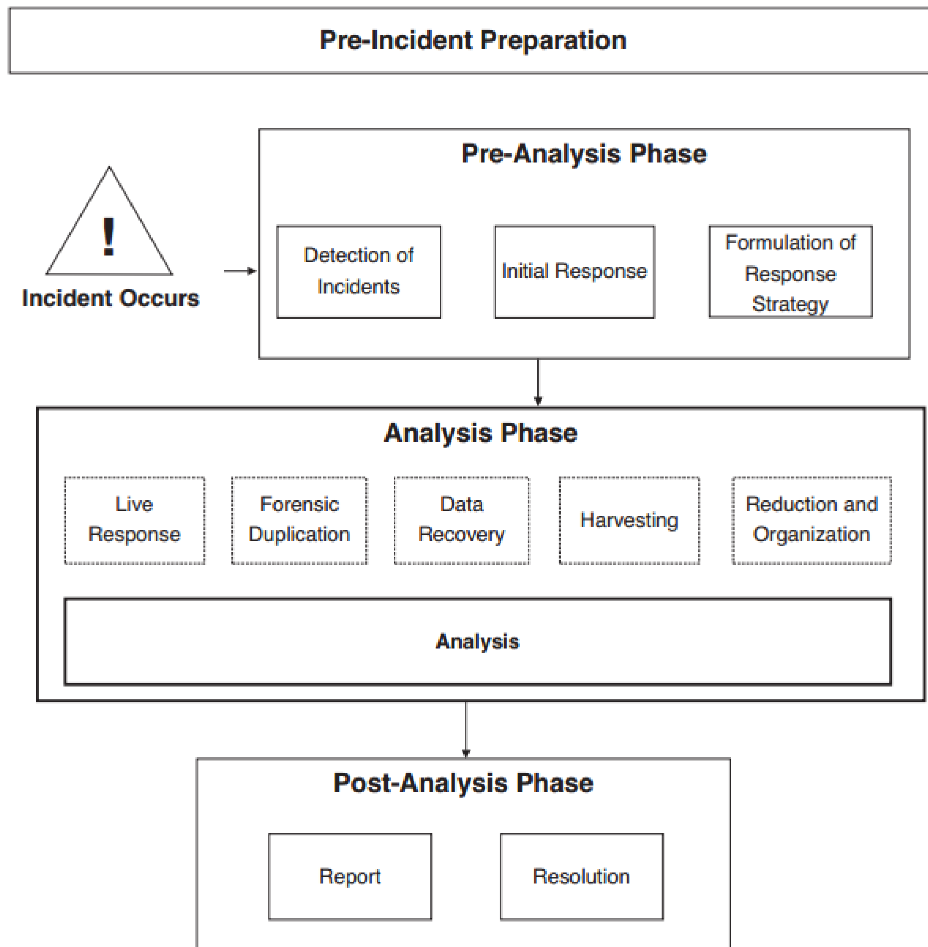

Figure 1. Common Process Model of Digital Forensics [1].

2.2 Forensic Analysis Technologies

This research focuses on the identification of hosts that have been involved in an incident. The significant challenge to timely and effective detection is the quantity of data that must be analyzed to properly identify the hosts of interest. There are many different methods of reducing forensic analyst workload that have been developed, each with varied success. Each method has different advantages and disadvantages over the other methods. This section covers some of the most common techniques used and the various pros and cons related to them.

2.2.1 Blacklist.

A blacklist is the straightforward method of identifying a file by hand and then adding the file to a known list of "bad" files [7]. The method of identifying the blacklisted files depends from system to system. Some of the most common methods are file location and name [8]. However, blacklists are rarely implemented without additional detection measures. This is because maintaining an up-to-date blacklist is nearly impossible [8]. If the malicious files that have been blacklisted are altered then the blacklist will not detect the now slightly different files. These mutations of the file allow it to remain undetected from the now-obsolete blacklist. Additionally, in order to keep up with these file alterations, a blacklist very quickly grows in size. This can become very resource intensive when scanning for blacklisted items.

2.2.2 Whitelist.

The opposite of the blacklist is the whitelist. This method of screening works by maintaining a list of approved files. The method of identifying approved files can vary depending on how the whitelist is implemented. A whitelist is most commonly used to identify trusted files that do not need to be checked by the forensic analyst [9].

However, whitelists have many of the same issues faced by blacklists. The whitelist must be updated whenever applications are updated or policies change [7]. Additionally, if a malicious file is added to the whitelist or similar enough to a file that has already been added, then it will be overlooked by the analyst.

2.2.3 Hashes.

When analyzing digital forensic data, a combination of a whitelist and blacklist is used to reduce the quantity of data. One of the most common methods is by using file hashes. In this manner, a whitelist of file hashes can remove the "known good" files and a blacklist of hashes can identify the "known bad" files. A hash is the resulting fixed length value returned when a file is processed by the hashing algorithm [10]. This hashing function produces a very different result even if only one bit has changed in the file being processed. This allows hashing to be a very effective method of ensuring that the file being inspected is matched exactly to the white or black list being used [2]. However, this causes the same problem encountered by other methods of implementing white or black lists; the file hash used for comparison must be updated every time the file is altered [11].

2.3 Related Research

Chwathe [12] was the first to propose the application of a whitelist to digital forensic data in 2009. In his work, he focused on the development of a hash-based whitelist to process the forensic data and return the items of interest. However his application faced the issues presented earlier. The hash function produces completely different hashes for files with very slight differences.

This field of research has since been continued by Breitinger, et al. [13] with the development of the `mvHash-B` algorithm. By using Similarity Preserving Hashing

(SPH), the `mvHash-B` algorith has made progress in recognizing when files have only been slightly changed. However, this research has only been applied to document files and JPG images. It also suffers from the requirement of a complete whitelist of hashes to match the files against.

2.4 Summary

This chapter introduces the field of Digital Forensics and covers the basic technologies used to reduce the data that must be analyzed. The application of whitelist and blacklist methods have been the foundation of research into reducing forensic analyst workload. The focus of related research has been on the application of hash-based whitelisting of files. This research expands the field of digital forensics by developing statistic-based whitelisting of processes.

III. Methodology

To address the challenge of ever increasing amounts of data to analyze, this research develops and demonstrates a method of reducing the analysis required through a statistic-based whitelisting of processes. Whitelisting in digital forensics has been shown to reduce the amount of data that must be reviewed by a forensic analyst [7]. This is done by comparing the processes of a host sample against the process in a whitelist. If the processes being compared reach an acceptable threshold, the process can be ignored as an acceptable process. By applying this concept to a large data set, a statistic whitelist can be generated without reviewing every process added to the whitelist.

This chapter presents this whitelisting system in the following sections. First, Section 3.1 covers how the whitelist benefits the forensic investigator and how the data is gathered. The structure and properties of the collected data are outlined in Section 3.2. The Pre-Processing, Build, and Testing functions of the whitelist are covered in Sections 3.3, 3.4, and 3.5 respectively. Lastly, Section 3.6 will summarize how the results of whitelist Testing are generated.

3.1 Whitelisting Overview

Whitelisting can be an effective method of filtering "known good" data in order to reduce the overall data collected. When applied to digital forensics, it can have a significant impact on the quantity of data that must be reviewed by an analyst. Traditionally, implementing a whitelist without verifying that the processes are good can have disastrous results [7]. If a malicious process is added to a whitelist, it will no longer be able to identify that process as malicious. Any host screened by said whitelist would identify the bad process as legitimate. This research takes advantage

of the repetition of processes in large data sets to circumvent these potentially adverse effects.

It is this repetition of processes across hosts of the same Operating System (OS) that is the basis for this research. Because of how each OS and program is initialized, many of the same processes will utilize identical resources when compared across multiple hosts. When a single identical process is compared across many hosts, each process will utilize the same resources. When a single version of the process utilizes different resources, or has additional connections, it is an indicator of potential forensic significance.

By generating a statistic-based whitelist from a large data set of process behavior, the whitelist can take advantage of this repetition of processes inherent to digital forensics. Hosts of the same OS often have the same processes running, accessing the same resources. By applying a statistic approach to these processes, a pattern emerges. If each host in the data set has the same process but only one is accessing different resources, it is a very clear indicator that an analyst should investigate said process as a potential item of interest. When this concept is expanded to the magnitude of several hundred hosts, the negative effect of adding a malicious process to the whitelist is mitigated because it will have a minor impact on the screening results.

3.2 Helix Data

The data used for this study is being generated by a suite of forensic incident response tools, specifically a plugin called Helix. The Helix plugin is not the same as the Helix3 incident response toolkit provided by E-Fense [14]. The data generated is placed in a MongoDB Database under a collection named helix.

MongoDB is an open-source database that uses key-value pairs to store data in a format-agnostic flat file. Each data entry is called a "document" and each document can contain additional subdocuments [15]. This results in a very flexible and fast platform for the storage of large quantities of data. The downside to these capabilities is that MongoDB queries are significantly more complicated than Structured Query Language (SQL) databases. The helix collection generated by the forensic tools has the document structure shown in Figure 2. Due to this unique structure, only a handful of the data fields collected have overlapping data fields. The lack of overlap means that the fields that can be connected are the primary data sources for this research, specifically the Processes, Modules, and TCP & UDP Connections.

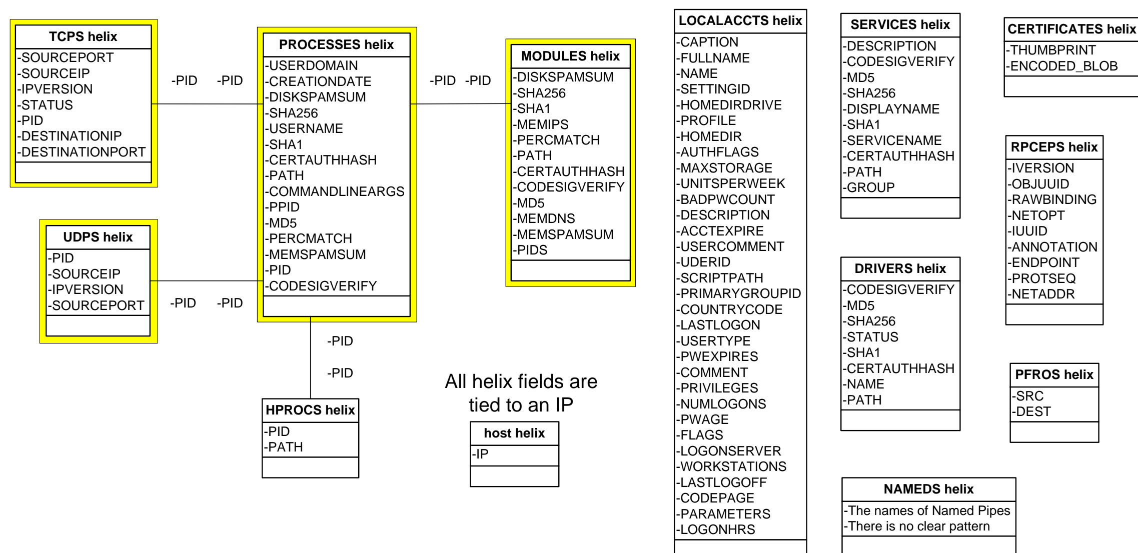

Figure 2. Helix Data Structure.

Processes, Modules, and the Transmission Control Protocol (TCP) & User Datagram Protocol (UDP) Connections are the only fields that have Process Identifier (PID) correlations. However, using the data from these fields requires more resources due to the unusual document structure generated by the Helix plugin. All of the data for each Host is contained within a single MongoDB document. Each of the Data fields shown in Figure 2 are contained within a single subdocument for each

Host document. In order to make the data easier to work with and to accomplish the research goal of not modifying the helix data directly, the data is pre-processed into a flat format. An example of the raw Helix data for a single process is shown in Figure 3.

```
PROCESSES
├─ PID               1252
├─ PPID              724
├─ PATH              C:\\WINDOWS\\system32\\svchost.exe
├─ USERNAME          LOCAL SERVICE
├─ USERDOMAIN        NT AUTHORITY
├─ COMMANDLINEARGS   C:\\WINDOWS\\system32\\svchost.exe -k LocalService
├─ MEMIPS
│   ├─ 9999::
│   └─ ::
├─ DISKSPAMSUM       96:q2uT2LP7gq7ukPsn5kQkxDOmk06t2U4CGHRQvnMa8AOEqiLmZ5vDAGJX
│                    DW/Ix:nusPz7b05iQC6UU4CGHRQEa8A3qiLmwS
├─ MEMSPAMSUM        96:q2uT2LP7gq7ukPsn5kQkxDOmk06t2U4CGHRQvnMa8AOEqiL6u5vDAGJX
│                    DW/Ix:nusPz7b05iQC6UU4CGHRQEa8A3qiL6rS
├─ CERTAUTHHASH      29D19562C49EEE6659DE92247AA8117AE043B849
├─ CODESIGVERIFY     Verified
├─ SHA1              49083AE3725A0488E0A8FBBE1335C745F70C4667
├─ SHA256            2910EBC692D833D949BFD56059E8106D324A276D5F165F874F3FB1B6C613CDD5
├─ MD5               27C6D03BCDB8CFEB96B716F3D8BE3E18
├─ PERCMATCH         97
└─ CREATIONDATE
    ├─ YEAR          2015
    ├─ MONTH         8
    ├─ DAY           17
    ├─ UTC           -4
    ├─ HOUR          3
    ├─ MINUTE        6
    └─ SECOND        58
```

Figure 3. Helix Raw Process Data.

The raw data shown in Figure 3 also demonstrates the three ways MongoDB stores key-value pairs. The first is the traditional one-to-one relationship where each key has a single value, as shown in `PID` and the majority of the other data fields. The second method of storage is the creation of a subdocument, where a key contains an array of key-value pairs. This is specifically shown by the `CREATIONDATE` data field. Lastly, a key can have an array of values, such as the multiple values stored under the `MEMIPS` key.

3.3 Pre-Processing

In order to break up the processing time for building and testing using the whitelist, the Helix data set is Pre-Processed to pull only the information used in representing the process behavior. One of the primary goals of this research is to avoid altering the original data set in order to avoid interfering with existing forensic tools that also use the data. This Pre-Processing utilizes the MongoDB Aggregation Pipeline to query the multiple sub-documents and pull only the information used to build the whitelist.

The MongoDB Aggregation pipeline is used to query elements from within a subdocument, through the execution of multiple stages. The Pre-Processing utilizes four of the stages possible in MongoDB 2.4: `Match`, `Unwind`, `Project`, and `Sort` [15]. `Match` functions like a traditional MongoDB query, but it can be used repeatedly in a single pipeline to continue to reduce the data being returned. `Unwind` is used to deconstruct an array field within a document; in this case it is used to directly query only the specific data fields from the overall Host document. The data being returned by the Aggregation pipeline is reduced during the `Project` stage. `Project` is used to limit the returned data to only the fields needed for the whitelist. Lastly, `Sort` is used to sort the returned data by a specified data field. Due to the processing needed for `Sort`, it is avoided as much as possible and only used when iterating through the Hosts of the Job ID being processed.

```
{"$match":{"job_id":"JOBID", "host":"HOSTIP"} },
{"$unwind":"$PROCESSES"},
{"$match":{"job_id":"JOBID", "host":"HOSTIP"} },
{"$project":{"_id":0, "PID":"$PROCESSES.PID",
     "PPID":"$PROCESSES.PPID",
     "PATH":{"$toUpper":      "$PROCESSES.PATH"} } }
```

Figure 4. Aggregation Pipeline Example.

The query shown in Figure 4 is used to find all Modules in the helix database for a specific Host. The first `Match` filters to find the single document that matches the Job ID and Host IP provided. The `Unwind` stage then breaks each `PROCESSES` subdocument out of the single document that was returned by the `Match` stage into individual documents. Another `Match` stage is performed to ensure the list of documents matches the query. This second `Match` is also where the query can be further narrowed down to focus on a single PID or element. Finally, the `Project` stage iterates through all the documents at this stage of the pipeline and filters the results to only return the desired values in a new document. In this query, only the `PID`, Parent Process Identifier (`PPID`), and `PATH` are returned.

The full Pre-Processing is straightforward in its execution. When it is executed, a Job ID is supplied and checked to ensure that the Job ID has not yet been processed. Then each Host IP recorded under the Job ID is sorted and iterated through. For each Host, the Processes, Modules, and TCP & UDP Connections are processed and saved to a new database. Because Path is the only reference to process locations in the Host file system, it is parsed and is recorded as the Location, Application or Module name, and the full Path. Additionally, during these conversions, the Drive letter is replaced to avoid later match issues. Figure 5 shows the resulting data structure once the Pre-Processing has completed. The Pre-Processing code is shown in Appendix A.

```
collection = Processed

Processed is used to track which JobID's/Hosts have been Pre-Processed
<tableName: Processed>
        <JobID          :UTF8>
        <HostIP         :UTF8>
        <loaded         :0|1>
        <completed      :0|1>

PreProcessList contains the records of each Process for each Host
<tableName: ProcessList>
        <JobID          :UTF8>
        <HostIP         :UTF8>
        <PID            :UTF8>
        <Application    :UTF8>
        <PPID           :UTF8>
        <Parent         :UTF8>
        <Path           :UTF8>
        <Location       :UTF8>
        <NumTCP         :INT32>
        <NumUDP         :INT32>

Modules contains the records of Modules associated with PreProcessList
<tableName: PreModules>
        <JobID          :UTF8>
        <HostIP         :UTF8>
        <PID            :UTF8>
        <Application    :UTF8>
        <DLL            :UTF8>
        <Path           :UTF8>
        <Location       :UTF8>

//PreTCPS contains the records of TCP connections associated with ProcessList
<tableName: PreTCPS>
        <JobID          :UTF8>
        <HostIP         :UTF8>
        <PID            :UTF8>
        <SourceIP       :UTF8>
        <SourcePort     :UTF8>
        <Version        :UTF8>

//PreUDPS contains the records of UDP connections associated with ProcessList
<tableName: PreUDPS>
        <JobID          :UTF8>
        <HostIP         :UTF8>
        <PID            :UTF8>
        <SourceIP       :UTF8>
        <SourcePort     :UTF8>
        <Version        :UTF8>
```

Figure 5. PreProcessed Data Structure.

3.4 Building the Whitelist

Once the Helix data has been Pre-Processed, the whitelist statistics are calculated. Building the whitelist focuses on iterating through all the Hosts that were scanned under a Job ID. For each host, the processes are evaluated and imported into the whitelist based on whether it is a new process or it matches a process already added to the whitelist. As part of the Build process, the whitelist keeps track of each host as it is imported. If for any reason the Build process fails or is interrupted in the middle of completing a Job ID, it can resume at the last host to be added to the whitelist. Figure 6 shows the logic flow diagram for the Build process.

The whitelist determines if a process has already been imported or not by assigning an Application ID (AppID) by comparing the Application name, Parent process, and Location. If the process being added to the whitelist does not match any previously imported processes then it is added to the whitelist. This simply involves copying each of the data fields from the Pre-Processed list into the whitelist format and inserting into the whitelist collection. A Quantity field with an initial value of 1 is also inserted with each record. This Quantity field will be incremented when this process is added to the whitelist when importing another host in order to track how often the process has appeared in the whitelist.

If the AppID of the process being added does match a previously imported process, then the Quantity field of each record is incremented by 1 for every record in the process being tested. This includes iterating through every Module and TCP & UDP Connection record for the process that matched the whitelist. If a Module or Connection record does not match any of the previous processes that matched the AppID, that new record will be added with a Quantity of 1. This allows the whitelist to recognize when a single process is an outlier from the other versions of that process that have been imported into the whitelist.

Because a Quantity field is maintained for ever record entered into the whitelist, anomalies can be identified when Testing against the whitelist. For example, a `CALC.EXE` process that is added to the whitelist 100 times only has a single instance which had an active TCP connection. By maintaining the Quantity record, when another `CALC.EXE` process that has an active TCP connection is Tested against the whitelist, it will recognize that while it is found in the WhiteList, it is an anomaly because of how rare it's occurrence rate is. Identifying these outliers will allow the whitelist to ignore the versions of these processes that are not of forensic interest. The whitelist code that contains the Build function can be found in Appendix B.

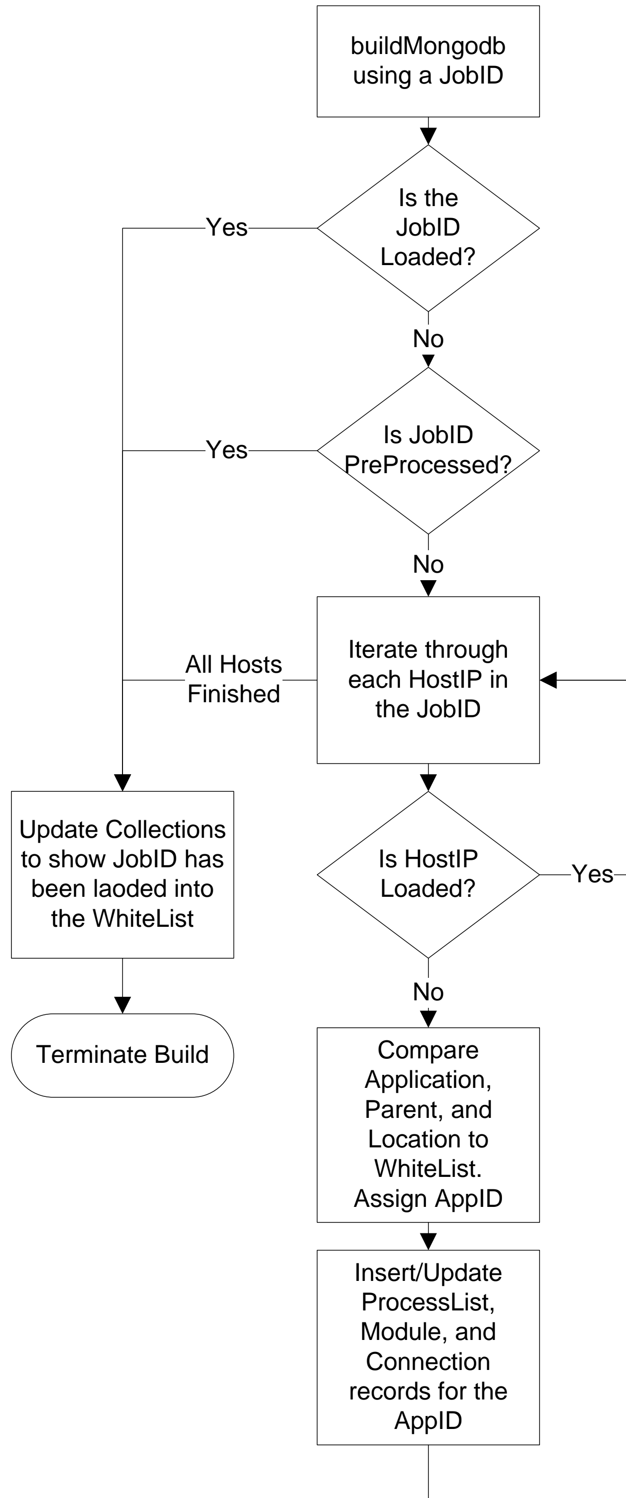

Figure 6. Build Logic Flow Diagram.

3.5 Testing Against the Whitelist

Once the whitelist has been built, a Job ID can be tested against the whitelist. Testing functions similarly to Build with the major difference in that scores are assigned to each Process record being Tested. When a process being Tested matches a process in the whitelist, a score is assigned by averaging how often each Module and Connection record appeared in the whitelist against the Quantity that the whitelist Process appeared in the whitelist. Thus, the process is being checked to see how often similar processes in the whitelist accessed the same resources as the process being Tested.

The Process is given an overall score based on a weighted average of the Module and Connection scores. The TCP Connections are given a weight of 20% of the Process score. Additionally, UDP Connections are given a weight of 20% as well. Lastly, the remaining 60% is assigned by the average Module score. These weights were based on previous unpublished research conducted at the Air Force Institute of Technology (AFIT) implementing whitelisting with very small numbers of hosts [16]. The adjustment of these weights and their implications are a subject of future research. Once each Process has been scored, a Host score is assigned based on the average of the Process scores for that Host.

Standard deviation and unweighted averages are recorded as well. Process score standard deviation is calculated based on the Module scores of Process. The Host score standard deviation is calculated using the Process scores of the Host. Figure 7 show the logic flow diagram for the Test process. The code containing the Testing function can be found in Appendix B.

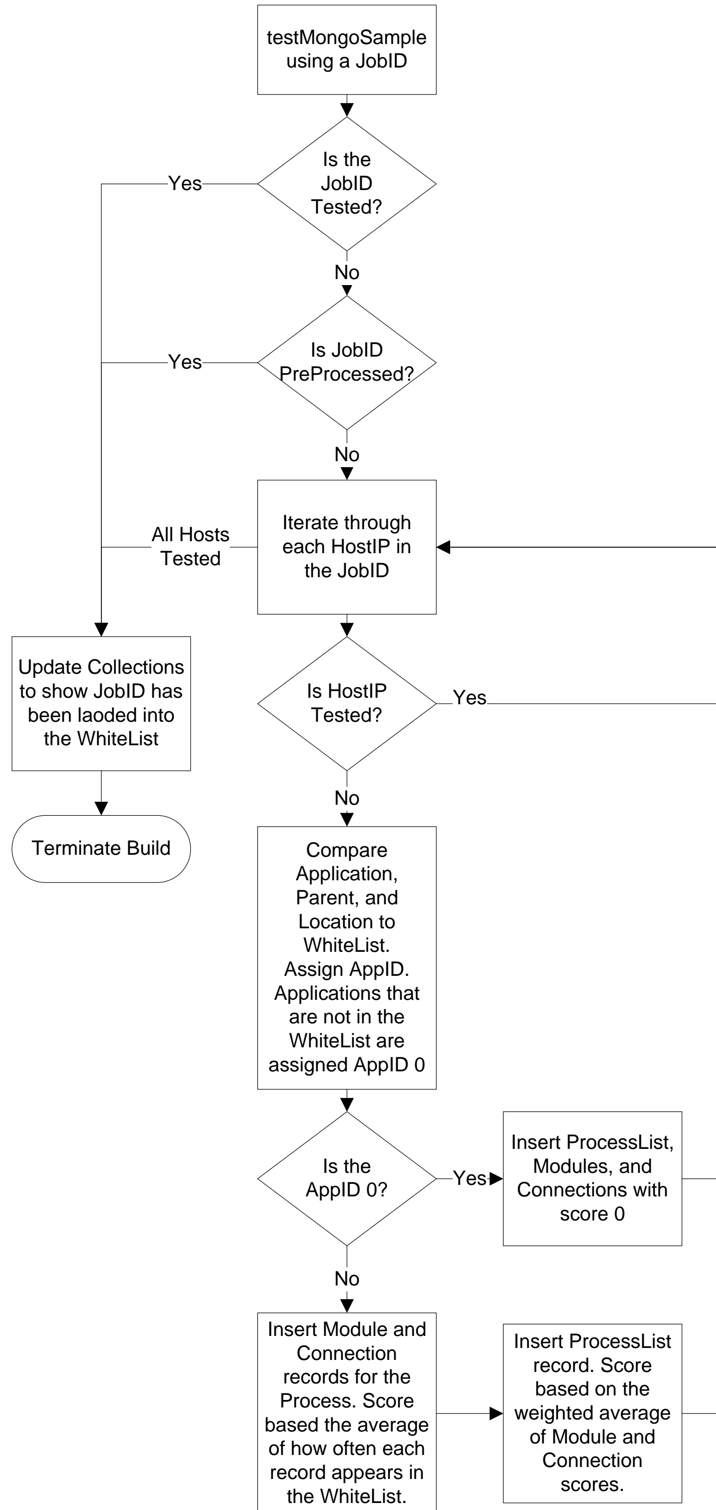

Figure 7. Test Logic Flow Diagram.

3.6 Generating Results

After a sample has been Tested, the Results can be generated. The Results process follows the logic flow diagram shown in Figure 8. First, the Job ID is checked to ensure that it has been Tested. Then, the Hosts are sorted based on the Host score assigned during Testing. For each Host, the Processes are sorted based on the Process score in order of lowest to highest. The Hosts and Processes with the lowest scores are given the highest priority for forensic analysis. Each Host is given a Host rank based on how low of a score it received. Each Process is then given a Process rank within each Host. For each Process, the Modules and Connection records are updated with the respective ranks as well. Once all Results have been generated, a script can be used to output the MongoDB Collection into a Comma-Separated Values (CSV) file format for investigator analysis. The code for Results generation is located in Appendix C. Additionally, the scripts used to extract the final CSV results can be found in Appendix D.

3.7 Summary

This chapter discusses the approach taken to generate a statistic whitelist in order to identify items of interest within large data sets. By using the inherent repetition of processes within large enterprise environments, a large set of unverified data can be used to generate a whitelist to identify common processes within the environment. The methods used to Process the data and Build the whitelist have been covered. Additionally, the Testing and generation of Results have been shown.

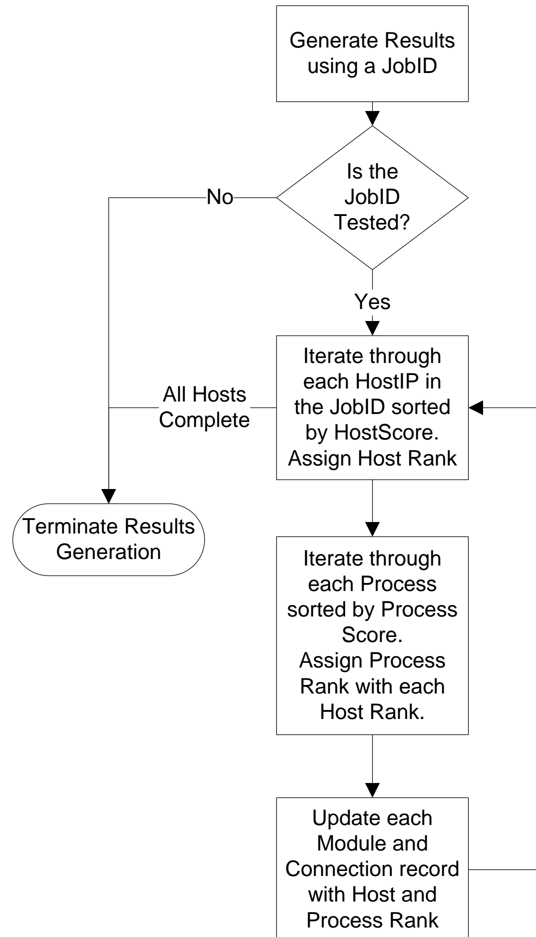

Figure 8. Results Logic Flow Diagram.

IV. Results and Analysis

This chapter outlines the results of the whitelist Testing. Once the whitelist has been Built and the data has been Tested, the Results must be generated. Results are generated based on the Host and Process scores assigned during Testing. These scores indicate how closely the data processes matched the process previously examined during the Build phase of the whitelist. By comparing the data against the whitelist, the processes that are the most unique will be assigned lower scores. This allows the forensic investigator to ignore the results with high scores and focus on the processes that are most likely to be identified as items of interest to the investigation.

In order to generate the results for this research, the whitelist was built using 512 unique hosts in the manner covered in Section 3.4. Second, a set of 519 new hosts were Tested against the whitelist according to the process covered in Section 3.5. Finally, the Results of the Test phase were generated as covered in Section 3.6. The Results were exported into human-readable form using the scripts shown in Appendix D.

4.1 Results

The Results generation ranks the Tested hosts according to the assigned HostScores as covered in Section 3.6. In order to evaluate the effectiveness of the whitelist, every process of the worst 5 hosts was examined. The five worst hosts are listed in Table 7 with the Host Score as well as the Standard Deviation of the Process Scores from the Host.

It was quickly determined that these hosts received the worst Host Scores because of the large number of processes that did not match processes contained in the whitelist. Of the 50 processes running on host 131.10.16.12, 17 received a score of 0 because they did not match the whitelist. Some of the processes are easily cleared of

Table 1. **Hosts with the Lowest Score.**

Host IP	Host Score	StDev of Process Scores
131.10.16.12	0.5145	0.4029
131.10.16.11	0.5927	0.4217
131.10.157.12	0.6555	0.3711
131.10.157.11	0.6958	0.3435
131.10.157.14	0.7323	0.3162

suspicion; the forensic incident response suit generates a new executable file whenever it collects data. Therefore, these randomly generated processes can be ignored because they were generated by the actions of the forensic analyst.

Additionally, there are a number of processes that have a different Parent processes than those that were added to the whitelist. By checking the Modules and Connections it becomes clear that these processes are not different than those contained in the whitelist. Finally, some processes are programs that were not installed on the hosts that were used to build the whitelist. The subsections below cover the specific processes of interest for each host.

4.1.1 #1: 131.10.16.12.

As the host with the lowest assigned Host score, there were a large number of processes that did not match the whitelist. Many of these were quickly ruled out. Three processes were part of the 64-bit version of McAfee antivirus, 3 had parent processes that were different then the whitelist, 2 were part of a digital network monitoring suite, 2 were caused by the Oracle server, and 1 was generated by the Java Quick Starter service.

However, there were some suspicious processes as well. A version of `WINLOGON.EXE` was listening on UDP port 1090 for the field bus message protocol. This is used to communicate with Industrial Control Systems (ICS) on the network. A more exact knowledge of the network is required to conclusively determine if this is unapproved

software or not. However, ICS control software should always be closely monitored as it is the bridge between types of networks.

Additionally, the `LSASS.EXE` process shows `RASSFM.DLL` as active. This is an item of further interest because `RASSFM.DLL` is only active when a password is being changed. When it is always active, it is a sign that a malicious attacker may be attempting to capture any password changes by using function hooking [17].

Table 2. Host #1: 131.10.16.12 Process Scores.

Total Processes	50
Score 1.0 - 0.76	17
Score 0.75 - 0.51	13
Score 0.5 - 0.1	3
Score 0.0	17
Suspicious Processes	2

4.1.2 #2: 131.10.16.11.

This host contains many similarities to host #1, to include the Java Quick Starter Service and the network monitoring suite. Additionally, the `LSASS.EXE` process on this host showed that `RASSFM.DLL` was active there as well. A new process of interest is `JAVAW.EXE`. This particular Java executable does not have a Graphical User Interface (GUI) and will execute in the background, unknown to the operator if they did not execute the process. Lastly, `WINLOGON.ESE` has an open connection to the Remote Assistance protocol port, port number 1053. This is a Windows service that allows remote control of the host over the network. While not necessarily malicious, such software must be closely monitored to prevent misuse.

Table 3. **Host #2: 131.10.16.11 Process Scores.**

Total Processes	69
Score 1.0 - 0.76	34
Score 0.75 - 0.51	11
Score 0.5 - 0.1	3
Score 0.0	21
Suspicious Processes	4

4.1.3 #3 - #5: 131.10.157.11, 131.10.157.12, and 131.10.157.14.

These three hosts are extremely similar to each other. The reason these hosts were scored so low becomes apparent very quickly. For each of these hosts, half of the un-matched processes belong to an information management program used to manage airfield automation. The other half of the unmatched processes belong to Automated battery backup system management software as well as remote host management suites.

These three hosts highlight the difficulty in creating a whitelist from unknown hosts. Within large enterprise networks, there is always a small subset of machines with special hardware and software installed. If some of these machines were included in the whitelist then the overall Host score for these hosts would be raised. However, this requires detailed knowledge of the hosts contained in the data before beginning the Build phase. This detailed knowledge was unavailable at the time of this research.

Table 4. **Host #3: 131.10.157.12 Process Scores.**

Total Processes	84
Score 1.0 - 0.76	40
Score 0.75 - 0.51	22
Score 0.5 - 0.1	4
Score 0.0	11
Suspicious Processes	2

Table 5. Host #4: 131.10.157.11 Process Scores.

Total Processes	75
Score 1.0 - 0.76	38
Score 0.75 - 0.51	22
Score 0.5 - 0.1	3
Score 0.0	17
Suspicious Processes	1

Table 6. Host #5: 131.10.157.14 Process Scores.

Total Processes	77
Score 1.0 - 0.76	42
Score 0.75 - 0.51	22
Score 0.5 - 0.1	5
Score 0.0	8
Suspicious Processes	2

4.1.4 Further Analysis.

The data from the five hosts with the lowest calculated scores show that 47% of the processes examined fall within the score range of 1.0 - 0.76. With even a cursory review of the remaining processes, greater than half of the processes that received a score of 0 can be ignored based on the process having never been imported into the whitelist. However, examination of these Tested hosts reveals that many of the lowest scoring hosts are not "normal" hosts. Many of these hosts were servers or had unusual hardware and software configurations. In order to evaluate the effectiveness of the whitelist, each host was examined in order of the Host Rank assigned by the whitelist. Hosts that were determined to be servers or unusual hardware and software configurations were eliminated as outliers. For example, the two hosts identified in Sections 4.1.1 and 4.1.2 were found to be servers and the next three shown in Section 4.1.3 are airfield automation and logistics computers with touchscreen devices. In order to better evaluate the effectiveness of the whitelist against "regular" host configurations, additional hosts were selected that were not servers or unusual hardware

configurations. The five hosts selected for additional analysis are listed in Table 7 with the Host Rank and Host Score as well as the Standard Deviation of the Process Scores from the Host.

Table 7. Additional Hosts Selected for Examination.

Host Rank	Host IP	Host Score	StDev of Process Scores
7	131.10.159.106	0.7558	0.2870
8	131.10.160.189	0.7579	0.2565
10	131.10.159.140	0.7666	0.2723
11	131.10.160.102	0.7670	0.2738
13	131.10.160.12	0.7734	0.2764

4.1.5 Host: 131.10.159.106.

For this host, there were seven processes that did not match the whitelist but many of these were quickly ruled out. Two of the processes were used for an external scanning device. Another three processes were part of the Microsoft Office Suite. The remaining processes included `WININIT.EXE` and `CSRSS.EXE` with a Parent process that had not been imported into the whitelist. Lastly, the process generated by the forensic tools was not in the whitelist. However, no processes were found to be items of interest.

Table 8. Host: 131.10.159.106 Process Scores.

Total Processes	99
Score 1.0 - 0.76	55
Score 0.75 - 0.51	31
Score 0.5 - 0.1	6
Score 0.0	7
Suspicious Processes	0

4.1.6 Host: 131.10.160.189.

Nothing unusual was discovered on this host. There were only four processes not already in the whitelist. Two were device management software for a document scanner. The remaining two processes were for Java and a USB Hub Controller. No processes of interest were found on this machine.

Table 9. Host: 131.10.160.189 Process Scores.

Total Processes	111
Score 1.0 - 0.76	65
Score 0.75 - 0.51	33
Score 0.5 - 0.1	9
Score 0.0	4
Suspicious Processes	0

4.1.7 Host: 131.10.159.140.

For this host 5 processes were not identified by the whitelist. One was Microsoft Powerpoint as well as one was generated by the forensic tools used to gather the data. Also, the remaining processes were used for a Brother brand printer management software and a scanner management process. No processes of interest were found on this machine.

Table 10. Host: 131.10.159.140 Process Scores.

Total Processes	98
Score 1.0 - 0.76	56
Score 0.75 - 0.51	29
Score 0.5 - 0.1	8
Score 0.0	5
Suspicious Processes	0

4.1.8 Host: 131.10.160.102.

Three instances of `CMD.EXE` were not identified by the whitelist. They were not identified because the Parent process had not been previously seen as the parent of a `CMD.EXE` process within the whitelist. There were only two additional processes that received a score of 0; the forensic process created during the data gathering processes and digital scanner management software. No processes of interest were found on this machine.

Table 11. Host: 131.10.160.102 Process Scores.

Total Processes	102
Score 1.0 - 0.76	60
Score 0.75 - 0.51	26
Score 0.5 - 0.1	11
Score 0.0	5
Suspicious Processes	0

4.1.9 Host: 131.10.160.12.

This host was identified because three of its unidentified processes were used to manage a digital label maker. The remaining processes that were not identified by the whitelist because one was generated by the forensic process and the final process was created during the installation of the Windows Configuration Manager. No processes of interest were found on this machine.

Table 12. Host: 131.10.160.12 Process Scores.

Total Processes	77
Score 1.0 - 0.76	55
Score 0.75 - 0.51	23
Score 0.5 - 0.1	10
Score 0.0	5
Suspicious Processes	0

4.1.10 Analysis Conclusions.

The five hosts with the lowest Host scores show that 47% of the processes examined fall within the score range of 1.0 - 0.76and with a cursory review of the remaining processes, greater than half of the processes that received a score of 0 can be ignored based on the process having never been imported into the whitelist. However, the whitelist identified these hosts with the lowest scores because they were very different from the hosts used to create the whitelist. These servers and machines with unique hardware configurations do not provide an adequate example of how the whitelist performs against hosts similar to a "normal" enterprise network host.

The data from the five additional "normal" hosts examined shows that 60% of the processes Tested against the whitelist fell within the score range of 1.0 - 0.76. Additionally, all of the processes that received a score of 0 were easily verified as not suspicious. This shows that the whitelist can effectively reduce the data that must be analyzed when identifying items of interest to a forensic incident.

4.2 Limitations

The methodology used in constructing the whitelist have some inherent limitations. As shown by the hosts analyzed in Section 4.1.3, if each host configuration is not included when building the whitelist, some hosts will receive deceivingly low scores. This can be mitigated by including each "version" of the hosts on the network when conducting the Build phase of the whitelist. However, a skilled forensic investigator familiar with the network being scanned can also identify these "variant" hosts quickly when analyzing the Results.

Also, the whitelist currently is unable to properly assign scores to OS processes that have multiple instances on a single host. In particular, `SVCHOST.EXE` and `WMIPRVSE.EXE`. These processes are created with the same process name, parent, and from the same

location as each other. However, each version of these processes performs different functions from each other and accesses different resources. The whitelist must be expanded to recognize the alternate versions of these processes to better score them. In the current implementation, nearly every process with a Process score between 0.01 - 0.75 is `SVCHOST.EXE` or `WMIPRVSE.EXE`. Improved analysis of these processes has the potential to improve the scores so that 79% of the Tested processes could be ignored without examination.

Additionally, 32-bit and 64-bit versions of the same process will be considered different processes and represented separately. This results in the whitelist having limited utility if the whitelist was built using 64-bit host processes but then tested against 32-bit processes, or vice-versa. This is mitigated by building the whitelist with a mix of both 64-bit and 32-bit OS and should be avoided by using a large data set to construct the whitelist.

If an Application is installed in a non-default file location then it will not match the same process installed in the default location due to how Process similarity is determined. This is mitigated by replacing the directory drive letter when Pre-Processing the application location and path but it does not prevent the machine user from installing software in non-default locations. Those processes will then be considered a "new" process due to the Assumptions reviewed in Chapter 1, Section 1.5.

4.3 Future Work

These limitations present very clear opportunities for future research. The whitelist can be upgraded to better distinguish between 32-bit and 64-bit versions of the same process. Additionally, the Pre-Processing presented in Section 3.3 can be expanded to recognize and ignore processes created by the forensic analyst in the process of gathering the data.

The Pre-Processing can also be expanded to better recognize process connections, in particular by differentiating between internal connection versus external connections. Distinguishing between well-known port numbers and randomly assigned port numbers could also be used to better evaluate the process connections. This would ensure that external connections on unexpected port numbers would very quickly be identified during Testing.

Finally, the whitelist must be expanded to recognize different versions of the same OS process. 32-bit and 64-bit processes often access the same resources but need to be tested separately to ensure the whitelist is only testing the same processes against each other. The method of scoring processes and hosts can also be expanded upon. Currently, the scoring does not take into account if a process was only imported into the whitelist a single time even if there are hundreds of hosts added to the whitelist. Initial applications of this scoring method caused all scores to settle to within 10% of the average score. A more effective implementation of such scoring will better identify processes that very rarely appear in the hosts added to the whitelist.

4.4 Summary

This chapter presents and analyzes the data collected in the demonstration of a behavior-based whitelisting of processes. By generating a whitelist from a large data set of unknown hosts, previously unexamined hosts can be examined more quickly. This research has shown that nearly half of the processes Tested by the whitelist can be immediately skipped by the analyst. When Testing process from the most common hosts of an enterprise network, the number of processes that must be inspected is reduced by 60%. While there is room for improvement, whitelisting of process based on the process behavior can help reduce the workload of processes that must be investigated by a forensic analyst.

V. Conclusions and Recommendations

This chapter summarizes the research performed in this study. The development of a statistic whitelist based on the inherent repetition of processes in an enterprise environment has been shown to be possible. Further improvements on this methodology can significantly improve a forensic analysts ability to identify hosts of interest during an incident response. Section 5.1 presents the conclusions reached during experimentation. Section 5.2 discusses the impact and contributions of this research.

5.1 Research Conclusion

The primary goal of this research, reducing the workload of forensic analysts through the development of a statistic-based whitelist, has been shown to have an impact. By using a large data set, it becomes apparent if a process is vastly different than the processes used to build the whitelist. Further improvements to this technique can make this very effective in identifying items of interest to the analyst. This technique has been shown to be a successful method of reducing analyst workload in identifying hosts of significance to an incident response.

5.2 Research Contributions

This research contributes to the DoD focus of better securing the IT infrastructure against malicious and criminal threats by providing a method of reducing the workload of forensic analysts. By minimizing the data that must be inspected, analysts can be better focus on the processes that pose the greatest potential of being a threat. This whitelist allows the analyst to use large enterprise network data instead of hand picking the hosts and processes to generate a whitelist which can successfully identify processes of interest to the forensic analyst during an incident response.

Appendix A. Code: Pre-Processing

1. preprocess.cpp

```
#include "preprocess.h"

#define DBNAME "thesis"

const bson_t *currentDoc;
mongoc_client_t *client;

int _tmain(int argc, TCHAR *argv[]) {

        uint32_t i;
        int ctr;

        TCHAR wprefix[1000], wjobID[1000];

        mongoc_collection_t *collection, *helixCollection;

        // Syntax: whitelist [process] dbname jobID(s)

        collection = NULL;
        helixCollection = NULL;

        mongoc_init ();

        client = mongoc_client_new ("mongodb://127.0.0.1/");

    //collection is the PreProcessed collection. This is where Process/Modules/
        ↪ Connections are processed from the helix collection
    //into simple key/value pairs to speed up the processing during build and
        ↪ test operations.
        collection = mongoc_client_get_collection (client, DBNAME, "Processed")
            ↪ ;
        if (collection == NULL) return 0;
        //helixCollection is the helix collection generated by Metasponse's
            ↪ helix module. This code never writes to this collection.
        helixCollection = mongoc_client_get_collection (client, DBNAME, "helix
            ↪ ");
```

```
        if (helixCollection == NULL) return 0;

        if (!_tcscmp(argv[1],"process"))
                printf("\nProcessing DB\n");

    printf("Using Database %s\n",DBNAME);

    //Process each JobID passed to the program.
        for (ctr = 3; ctr < argc; ctr++) {
                for (i = 0; i < _tcslen(argv[ctr]); i++) wprefix[i] = argv[ctr
                    ↪ ][i]; wprefix[_tcslen(argv[ctr])] = 0;
                _tcscpy(wjobID,wprefix);
                if (!_tcscmp(argv[1],"process")) {
                        if (!processMongoSample(collection, helixCollection,
                            ↪ wjobID)) return 0;
                } else
                        printf("\nBad Operation Specified\nwhitelist process
                            ↪ database JobID(s)\n");
        }
    printf("Exiting\n");

    //Clean up
    mongoc_collection_destroy (collection);
    mongoc_collection_destroy (helixCollection);
        mongoc_client_destroy (client);
        mongoc_cleanup ();

        return 1;

} //END main()

uint8_t processMongoSample(mongoc_collection_t *collection, mongoc_collection_t
    ↪ *helixCollection, TCHAR *jobID) {

        char *location, *location2;
        int rc, tcpI, udpI;
        const bson_t *doc, *doc2, *doc3, *pipeline, *pipeline2, *hostpipeline,
            ↪ *query, *currentHost, *hostdoc;
        mongoc_cursor_t *importCursor, *importCursor2, *hostCursor, *cursor;
```

```
    char *applicationS, *parentS, *dllS, *pathS, *newpathS, *pathS2, *
        ↪ newpathS2, *hostIP, *str;
    char *pidI, *ppidI, *portI;
    char *versionS, *connectipS;
    bool debugit = false;

    bson_error_t t_error;
    t_error.domain = 0;
    t_error.code = 0;

//Check if the jobID has already been processed
    query = BCON_NEW ( "$query", "{", "tableName", "Processed", "JobID",
        ↪ BCON_UTF8(jobID), "completed", BCON_INT32(1), "}");
    cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0, 0,
        ↪ 0, query, NULL, NULL);
    if (mongoc_cursor_next(cursor, &doc)) {
            printf("JobID Already Loaded\n");
            mongoc_cursor_destroy(cursor);
            return 1; //If the jobID has already been processed then exit
    } else {
        if (debugit) {printf("JobID Not Loaded\n");}
    }

printf("Iterating through all Hosts for Job ID %s\n",jobID);

//The Host IP are iterated through after sorting. If the program ever
    ↪ crashes before completing test, then it will only lose the progress
//for the Host IP it crashed on. Re-running the same command will start
    ↪ testing from the same Host IP it crashed on. (This may result in
//a handful of Processes/Modules/Connections being processed twice.
    ↪ Provided many Hosts are being processed, this should have a very
    ↪ minor impact.

hostpipeline = BCON_NEW ("pipeline", "[",
                    "{", "$match", "{", "job_id", BCON_UTF8(jobID), "}",
                        ↪ "}",
                    "{", "$project", "{", "_id", BCON_INT32(0), "HOST", "
                        ↪ $host", "}", "}",
                    "{", "$sort", "{", "HOST", BCON_INT32(1), "}", "}",
                    "]");
```

```
hostCursor = mongoc_collection_aggregate (helixCollection,MONGOC_QUERY_NONE
    ↪ , hostpipeline ,NULL,NULL);

if (mongoc_cursor_error(hostCursor,&t_error))
    printf("\tmongoc_collection_aggregate     error = %d.%d: %s\n",t_error.
        ↪ domain,t_error.code,t_error.message);

//Loop through each Host IP
while (mongoc_cursor_next(hostCursor, &currentHost)) {
    if (debugit) { str = bson_as_json (currentDoc, NULL); printf("Hosts\t\
        ↪ tstr = %s\n",str); bson_free (str); }
    getMongoValue(currentHost, "HOST", (void **) &hostIP);

    query = BCON_NEW ( "$query", "{", "tableName", "Processed", "JobID",
        ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "loaded",
        ↪ BCON_INT32(1), "}");
    cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0, 0,
        ↪ 0, query, NULL, NULL);

    if (mongoc_cursor_next(cursor, &hostdoc)) {
        printf("Host IP %s Already Processed\n",hostIP);
        mongoc_cursor_destroy(cursor);
        //If the hostIP has already been loaded then continue to the next
            ↪ host
    } else {
        if (debugit) { printf("Checking ProcessList and Module records\n");
            ↪ }
        printf("%s Checking ProcessList and Module records\n",hostIP);

        pipeline = BCON_NEW ("pipeline", "[",
                        "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                            ↪ "host", BCON_UTF8(hostIP), "}", "}",
                        "{", "$unwind", "$PROCESSES", "}",
                        "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                            ↪ "host", BCON_UTF8(hostIP), "}", "}",
                        "{", "$project", "{", "_id", BCON_INT32(0), "
                            ↪ PID", "$PROCESSES.PID", "PPID", "
                            ↪ $PROCESSES.PPID", "PATH", "{", "$toUpper
                            ↪ ", "$PROCESSES.PATH", "}", "}", "}",
```

```
                    "]");

importCursor = mongoc_collection_aggregate (helixCollection,
    ↪ MONGOC_QUERY_NONE, pipeline,NULL,NULL);

if (mongoc_cursor_error(importCursor,&t_error))
    printf("\tmongoc_collection_aggregate     error = %d.%d: %s\n",
        ↪ t_error.domain, t_error.code, t_error.message);

//Iterate through each Process
while ( mongoc_cursor_next(importCursor, &currentDoc) )  {
    if (debugit) { str = bson_as_json (currentDoc, NULL); printf("
        ↪ Processes\tstr = %s\n",str); bson_free (str); }
    getMongoValue(currentDoc, "PID", (void **) &pidI);
    getMongoValue(currentDoc, "PPID", (void **) &ppidI);
    parentS = getMongoApp(helixCollection,jobID,hostIP,ppidI);
    if (parentS == NULL) {
        if (debugit) { printf("parentS NULL in PPID %s\n",ppidI); }
        parentS = (char *) malloc(sizeof(char) * 4);
        sprintf(parentS,"N/A");
    }
    getMongoValue(currentDoc, "PATH", (void **) &pathS);
    if (pathS == NULL) {
        if (debugit) { printf("pathS NULL in PID %s\n",pidI); }
        pathS = null();
    }
    newpathS = convert_path(pathS);
    location = convert_loc(newpathS);
    applicationS = convert_app(pathS);

    doc = BCON_NEW ("tableName", "PreProcessList", "JobID",
        ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
                    "PID", BCON_UTF8(pidI), "Application",
                        ↪ BCON_UTF8(applicationS), "PPID",
                        ↪ BCON_UTF8(ppidI),
                    "Parent", BCON_UTF8(parentS), "Location",
                        ↪ BCON_UTF8(location), "Path",
                        ↪ BCON_UTF8(newpathS)
                );
```

```
rc = mongoc_collection_insert( collection , MONGOC_INSERT_NONE,
    ↪ doc , NULL, NULL);

//Start Modules Processing
if (debugit) { printf("Starting Modules\n"); }

pipeline2 = BCON_NEW ("pipeline", "[",
                "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                    ↪  "host", BCON_UTF8(hostIP), "}", "}",
                "{", "$unwind", "$MODULES", "}",
                "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                    ↪  "host", BCON_UTF8(hostIP), "MODULES.
                    ↪ PIDS", BCON_UTF8(pidI), "}", "}",
                "{", "$project", "{", "_id", BCON_INT32(0), "
                    ↪ PATH", "{", "$toUpper", "$MODULES.PATH",
                    ↪  "}", "}", "}",
                "]");

importCursor2 = mongoc_collection_aggregate (helixCollection ,
    ↪ MONGOC_QUERY_NONE, pipeline2 ,NULL,NULL);

if (mongoc_cursor_error(importCursor2,&t_error))
    printf("\tmongoc_collection_aggregate    error = %d.%d: %s
        ↪ \n",t_error.domain,t_error.code,t_error.message);

while ( mongoc_cursor_next(importCursor2, &currentDoc) )  {
    if (debugit) { str = bson_as_json (currentDoc, NULL);
        ↪ printf("Modules\t\tstr = %s\n",str); bson_free (str)
        ↪ ; }
    getMongoValue(currentDoc, "PATH", (void **) &pathS2);
    if (pathS2 == NULL) {
        if (debugit) { printf("pathS2 NULL in PID %s module\n",
            ↪ pidI); }
        pathS2 = null();
    }
    newpathS2 = convert_path(pathS2);
    location2 = convert_loc(newpathS2);
    dllS = convert_app(pathS2);
```

```
        doc2 = BCON_NEW ("tableName", "PreModules", "JobID",
            ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
                        "PID", BCON_UTF8(pidI), "Application",
                            ↪ BCON_UTF8(applicationS), "DLL",
                            ↪ BCON_UTF8(dllS),
                        "Location", BCON_UTF8(location2), "Path
                            ↪ ", BCON_UTF8(newpathS2)
                    );

        rc = mongoc_collection_insert( collection,
            ↪ MONGOC_INSERT_NONE, doc2, NULL, NULL);

    }//End Modules Loop

    pipeline2 = BCON_NEW ("pipeline", "[",
                "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                    ↪  "host", BCON_UTF8(hostIP), "}", "}",
                "{", "$unwind", "$TCPS", "}",
                "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                    ↪  "host", BCON_UTF8(hostIP), "TCPS.PID",
                    ↪ BCON_UTF8(pidI), "}", "}",
                "{", "$project", "{", "_id", BCON_INT32(0), "
                    ↪ SOURCEPORT", "$TCPS.SOURCEPORT", "}",
                    ↪ "}",
                "]");

    importCursor2 = mongoc_collection_aggregate (helixCollection,
        ↪ MONGOC_QUERY_NONE, pipeline2 ,NULL,NULL);

    if (mongoc_cursor_error(importCursor2,&t_error))
        printf("\tmongoc_collection_aggregate     error = %d.%d: %s
            ↪ \n",t_error.domain,t_error.code,t_error.message);

    if (debugit) { printf("Starting TCP Connection Count Loop\n");
        ↪ }
    tcpI = 0;
    while ( mongoc_cursor_next(importCursor2, &currentDoc) ) {
        if (debugit) { str = bson_as_json (currentDoc, NULL);
            ↪ printf("TCP Connections\t\tstr = %s\n",str);
            ↪ bson_free (str); }
```

```
    tcpI++;
}//End TCP Connections Loop

pipeline2 = BCON_NEW ("pipeline", "[",
                "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                    ↪  "host", BCON_UTF8(hostIP), "}", "}",
                "{", "$unwind", "$UDPS", "}",
                "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                    ↪  "host", BCON_UTF8(hostIP), "UDPS.PID",
                    ↪ BCON_UTF8(pidI), "}", "}",
                "{", "$project", "{", "_id", BCON_INT32(0), "
                    ↪ SOURCEPORT", "$UDPS.SOURCEPORT", "}",
                    ↪ "}",
                "]");

importCursor2 = mongoc_collection_aggregate (helixCollection,
    ↪ MONGOC_QUERY_NONE, pipeline2 ,NULL,NULL);

if (mongoc_cursor_error(importCursor2,&t_error))
    printf("\tmongoc_collection_aggregate     error = %d.%d: %s
        ↪ \n",t_error.domain,t_error.code,t_error.message);

if (debugit) { printf("Starting UDP Connection Count Loop\n");
    ↪ }
udpI = 0;
while ( mongoc_cursor_next(importCursor2, &currentDoc) ) {
    if (debugit) { str = bson_as_json (currentDoc, NULL);
        ↪ printf("UDP Connections\t\tstr = %s\n",str);
        ↪ bson_free (str); }
    udpI++;
}//End UDP Connections Loop

if (debugit) { printf("NumTCP %i\tNumUDP %i\n",tcpI,udpI); }
doc2 = BCON_NEW ("tableName", "PreProcessList", "JobID",
    ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
                        "PID", BCON_UTF8(pidI) );
doc3 = BCON_NEW ("$inc", "{", "NumTCP", BCON_INT32(tcpI), "
    ↪ NumUDP", BCON_INT32(udpI), "}");
if (!mongoc_collection_update( collection, MONGOC_UPDATE_UPSERT
    ↪ , doc2, doc3, NULL, &t_error))
```

```
            printf("Update Error: %i.%i:\t%s\n",t_error.domain,
              ↪ t_error.code,t_error.message);

    }//End Processes Loop

    if (debugit) { printf("Checking TCP records\n"); }

    pipeline = BCON_NEW ("pipeline", "[",
                    "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                      ↪  "host", BCON_UTF8(hostIP), "}", "}",
                    "{", "$unwind", "$TCPS", "}",
                    "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                      ↪  "host", BCON_UTF8(hostIP), "}", "}",
                    "{", "$project", "{", "_id", BCON_INT32(0), "
                      ↪ PID", "$TCPS.PID", "SOURCEPORT", "$TCPS.
                      ↪ SOURCEPORT",
                                "SOURCEIP", "$TCPS.SOURCEIP", "
                                  ↪ IPVERSION", "$TCPS.
                                  ↪ IPVERSION",
                                "}", "}",
                    "]");

    importCursor = mongoc_collection_aggregate (helixCollection,
       ↪ MONGOC_QUERY_NONE, pipeline,NULL,NULL);

    if (mongoc_cursor_error(importCursor,&t_error))
        printf("\tmongoc_collection_aggregate    error = %d.%d: %s\n",
           ↪ t_error.domain,t_error.code,t_error.message);

    if (debugit) { printf("Before TCP Loop\n"); }

    //Iterate through all the TCP Connections being tested
    while ( mongoc_cursor_next(importCursor, &currentDoc) )  {
        if (debugit) { str = bson_as_json (currentDoc, NULL); printf("
           ↪ TCPS\t\tstr = %s\n",str); bson_free (str); }
        getMongoValue(currentDoc, "PID", (void **) &pidI);
        getMongoValue(currentDoc, "SOURCEPORT", (void **) &portI);
        getMongoValue(currentDoc, "SOURCEIP", (void **) &connectipS);
        if (connectipS == NULL) {
            connectipS = null();
```

```
        }
        getMongoValue(currentDoc, "IPVERSION", (void **) &versionS);
        if (versionS == NULL) {
            versionS = null();
        }

        doc = BCON_NEW ("tableName", "PreTCPS", "JobID", BCON_UTF8(
            ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                            "PID", BCON_UTF8(pidI), "SourcePort",
                                ↪ BCON_UTF8(portI)
                        ,"SourceIP", BCON_UTF8(connectipS), "Version",
                            ↪ BCON_UTF8(versionS)
                        );

        rc = mongoc_collection_insert( collection, MONGOC_INSERT_NONE,
            ↪ doc, NULL, NULL);

    }//End TCP Loop

    if (debugit) { printf("Checking UDP records\n"); }

    pipeline = BCON_NEW ("pipeline", "[",
                        "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                            ↪  "host", BCON_UTF8(hostIP), "}", "}",
                        "{", "$unwind", "$UDPS", "}",
                        "{", "$match", "{", "job_id", BCON_UTF8(jobID),
                            ↪  "host", BCON_UTF8(hostIP), "}", "}",
                        "{", "$project", "{", "_id", BCON_INT32(0), "
                            ↪ PID", "$UDPS.PID", "SOURCEPORT", "$UDPS.
                            ↪ SOURCEPORT",
                                        "SOURCEIP", "$UDPS.SOURCEIP", "
                                            ↪ IPVERSION", "$UDPS.
                                            ↪ IPVERSION",
                                        "}", "}",
                        "]");

    importCursor = mongoc_collection_aggregate (helixCollection,
        ↪ MONGOC_QUERY_NONE, pipeline,NULL,NULL);

    if (mongoc_cursor_error(importCursor,&t_error))
```

```
        printf("\tmongoc_collection_aggregate     error = %d.%d: %s\n",
            ↪ t_error.domain,t_error.code,t_error.message);

    if (debugit) { printf("Before UDP Loop\n"); }

    //Iterate through all the UDP Connections being tested
    while ( mongoc_cursor_next(importCursor, &currentDoc) )  {
        if (debugit) { str = bson_as_json (currentDoc, NULL); printf("
            ↪ UDPS\t\tstr = %s\n",str); bson_free (str); }
        getMongoValue(currentDoc, "PID", (void **) &pidI);
        getMongoValue(currentDoc, "SOURCEPORT", (void **) &portI);
        getMongoValue(currentDoc, "SOURCEIP", (void **) &connectipS);
        if (connectipS == NULL) {
            connectipS = null();
        }
        getMongoValue(currentDoc, "IPVERSION", (void **) &versionS);
        if (versionS == NULL) {
            versionS = null();
        }

        doc = BCON_NEW ("tableName", "PreUDPS", "JobID", BCON_UTF8(
            ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                            "PID", BCON_UTF8(pidI), "SourcePort",
                                ↪ BCON_UTF8(portI)
                        ,"SourceIP", BCON_UTF8(connectipS), "Version",
                            ↪ BCON_UTF8(versionS)
                        );

        rc = mongoc_collection_insert( collection, MONGOC_INSERT_NONE,
            ↪ doc, NULL, NULL);

    }//End UDP Loop



    //Update HostIP as loaded
    doc = BCON_NEW ("tableName", "Processed", "JobID", BCON_UTF8(jobID)
        ↪ , "HostIP", BCON_UTF8(hostIP), "loaded", BCON_INT32(1) );
    rc = mongoc_collection_insert(  collection, MONGOC_INSERT_NONE, doc
        ↪ , NULL, &t_error);
```

```
            if (debugit) printf("mongoc_collection_insertrc = %i      error = %d
                ↪ .%d: %s\n",rc,t_error.domain,t_error.code,t_error.message);

            printf("Host IP %s Processed\n", hostIP);
        }//End IF for HostIP not loaded

    }//End JobID Loop
    mongoc_cursor_destroy(importCursor);
    mongoc_cursor_destroy(importCursor2);
    mongoc_cursor_destroy(hostCursor);

    //Mark the JobID as completed
    doc = BCON_NEW ("tableName", "Processed", "JobID", BCON_UTF8(jobID), "
        ↪ completed", BCON_INT32(1) );
    rc = mongoc_collection_insert( collection, MONGOC_INSERT_NONE, doc, NULL, &
        ↪ t_error);
    if (debugit) printf("mongoc_collection_insertrc = %i      error = %d.%d: %s\
        ↪ n",rc,t_error.domain,t_error.code,t_error.message);

    if (debugit) { printf("rc = %d\n",rc); }
        return 0;

} //END processMongoSample()

/***************************************************************************/
// Convert the string to upper case and return a new string
char *strtoupper_c (const char *lower) {
        unsigned int i;
        char *upper;
        if (lower == NULL) return NULL;
        upper = (char *) malloc(sizeof(char) * (strlen(lower) + 1));
        for (i = 0; i < strlen(lower); i++)     upper[i] = toupper(lower[i]);
        upper[strlen(lower)] = 0;
        return upper;
}

// Convert the string to upper case in place (i.e., do not return a new string)
void strtoupperIP (CHAR *lower) {
        unsigned int i;
        if (lower == NULL) return;
```

```
        for (i = 0; i < strlen(lower); i++)      lower[i] = toupper(lower[i]);
}

/**************************************************************************/
//convert_path removes the drive letter of the path and replaces it with 'ROOT'
char *convert_path(char *path) {
        char *basename, *newpath;
        int i, len;
        basename = strrchr(path,'\\');
        if (basename == NULL) {
        //There were no "\\" characters in path. This means Path is null or
            ↪ helix only returned the application/module name
        return path;
        } else {
        len = strlen(path);
        for (i = 0; i < ( len -1 ); i++) path[i] = path[i+1];
        path[len-1] = 0;
        newpath = (char *) malloc(sizeof(char) * (strlen(path) + 5));
        memcpy( newpath, "ROOT", 4 );
        memcpy( newpath + 4, path, strlen(path) );
        newpath[strlen(path)+4] = 0;
        return newpath;
        }
        printf("\n\nSomething happened in convert_path\n\n");
        return path;
} //END convert_path

// convert_loc takes a process/module path and removes the process/module to
    ↪ provide the directory path to said process/module
char *convert_loc(char *path) {
        char *basename, *location;
        int i;
        basename = strrchr(path,'\\');
        if (basename == NULL) {
        //There were no "\\" characters in path. This means Path is null or
            ↪ helix only returned the application/module name
        if (strcmp(path, "SYSTEM IDLE PROCESS") == 0) {
            location = (char *) malloc(sizeof(char) * 20);
            sprintf(location,"SYSTEM IDLE PROCESS");
            return location;
```

```
        } else if (strcmp(path, "SYSTEM") == 0) {
            location = (char *) malloc(sizeof(char) * 7);
            sprintf(location,"SYSTEM");
            return location;
        } else {
            return null();
        }
        } else {
            //Copy the Location from the Path (everything but the application/
                ↪ module name)
        location = (char *) malloc(sizeof(char) * (basename - path + 2));
        for (i = 0; i < (basename - path + 1); i++) location[i] = path[i];
            ↪ location[i] = 0;
        return location;
        }
        printf("\n\nSomething happed in convert_loc\n\n");
        return null();
} //END convert_loc

// convert_app takes a process/module path and removes the directory path to
    ↪ provide the process/module name
char *convert_app(char *path) {
        char *name;
        name = strrchr(path,'\\');
        if (name == NULL) {
        //There were no "\\" characters in path. This means Path is null or
            ↪ helix only returned the application/module name
        if (strcmp(path, "SYSTEM IDLE PROCESS") == 0) {
            name = (char *) malloc(sizeof(char) * 20);
            sprintf(name,"SYSTEM IDLE PROCESS");
            return name;
        } else if (strcmp(path, "SYSTEM") == 0) {
            name = (char *) malloc(sizeof(char) * 7);
            sprintf(name,"SYSTEM");
            return name;
        } else {
            if (path == NULL) {
                //helix recorded nothing for Path
                return null();
            } else {
```

```
        //helix only recorded the application/module name for Path
        return path;
    }
    printf("\n\nSomething happed in convert_app null check\n\n");
    return null();
  }
  } else {
    //Return the application/module name incremented by 1 to remove the
        ↪  '\\'
  name++;
  return name;
  }
  printf("\n\nSomething happed in convert_app\n\n");
  return null();
} //END convert_app

char *null() {
  char *retval;
  retval = (char *) malloc(sizeof(char) * 5);
  sprintf(retval,"NULL");
  return retval;
} //END null

/****************************************************************************/
// getMongoValue returns the value of the inputKey in the imported document
void getMongoValue(const bson_t *doc, const char *inputKey, void **outputValue)
  ↪  {
  char *str, *retval, *wkey, *inputKeyUpper;
  bson_iter_t iter;
  const bson_value_t *bvalue;
  int i;
  int32_t *indirectInt32;

  inputKeyUpper = strtoupper_c(inputKey);
  str = bson_as_json (doc, NULL);
  if (bson_iter_init(&iter, doc)) {
    while (bson_iter_next(&iter)) {
        wkey = (char *) malloc(sizeof(char) * (strlen(bson_iter_key
          ↪ (&iter)) + 1));
        strcpy(wkey, bson_iter_key (&iter));
```

```
                    strtoupperIP(wkey);
                    if (!strcmp(wkey,inputKeyUpper)) {
                            bvalue = bson_iter_value (&iter);
                            if (bvalue->value_type == BSON_TYPE_UTF8) {
                                    retval = (char *) malloc(sizeof(char) * (
                                        ↪ bvalue->value.v_utf8.len + 1));
                                    for (i = 0; i < (int) bvalue->value.v_utf8.
                                        ↪ len; i++) retval[i] = bvalue->value.
                                        ↪ v_utf8.str[i];
                                    retval[i] = 0;
                                    free(wkey);
                                    free(inputKeyUpper);
                                    bson_free (str);
                                    *outputValue = (void *) retval;
                            } else if (bvalue->value_type == BSON_TYPE_INT32) {
                                    free(wkey);
                                    free(inputKeyUpper);
                                    bson_free (str);
                                    indirectInt32 = (int32_t *) outputValue;
                                    *indirectInt32 = bvalue->value.v_int32;
                            } else {
                                    printf("**getValue** key = %s\t\ttype = %i\
                                        ↪ ndocument = %s\n\n",inputKey,bvalue->
                                        ↪ value_type,str);
                            }
                            return;
                    }
                    free(wkey);
            }
    }

        free(inputKeyUpper);
        bson_free (str);

        return;
} //END getMongoValue

// getMongoApp returns the name of an Application using the PID, JobID, and
    ↪ HostIP, used to determine PPID Application Name
```

```
char * getMongoApp(mongoc_collection_t *helixCollection, TCHAR *jobID, TCHAR *
    ↪ hostIP, TCHAR *pid) {
        char *retval, *pathS;
        const bson_t *doc, *pipeline;
    mongoc_cursor_t *cursor;
    bson_error_t t_error;

        t_error.domain = 0;
        t_error.code = 0;

    pipeline = BCON_NEW ("pipeline", "[",
                        "{", "$match", "{", "job_id", BCON_UTF8(jobID), "host",
                            ↪ BCON_UTF8(hostIP), "}", "}",
                        "{", "$unwind", "$PROCESSES", "}",
                        "{", "$match", "{", "job_id", BCON_UTF8(jobID), "host",
                            ↪ BCON_UTF8(hostIP), "PROCESSES.PID", BCON_UTF8(
                            ↪ pid), "}", "}",
                        "{", "$project", "{", "_id", BCON_INT32(0), "PATH",
                            ↪ "{", "$toUpper", "$PROCESSES.PATH", "}", "}",
                            ↪ "}",
                        "]");

    cursor = mongoc_collection_aggregate (helixCollection, MONGOC_QUERY_NONE,
        ↪ pipeline, NULL, NULL);
    if (mongoc_cursor_error(cursor,&t_error))
            printf("\tmongoc_collection_aggregate    error = %d.%d: %s\n",
                ↪ t_error.domain,t_error.code,t_error.message);

        if (mongoc_cursor_next(cursor, &doc)) {
                getMongoValue(doc, "PATH", (void **) &pathS);
        retval = convert_app(pathS);
        mongoc_cursor_destroy (cursor);
        return retval;
        }
    retval = null();
    return retval;
} //END getMongoApp
```

2. preprocess.h

```
#include <math.h>

/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <stdint.h>
#include <string.h>

#define _tmain main
#define TRUE true
#define FALSE false

#define TCHAR char
#define CHAR char

#define _tcslen strlen
#define _tcscmp strcmp
#define _tcscpy strcpy
#define _tcscat strcat
#define _totupper toupper

#include <mongoc.h>

#ifdef _UNICODE
#define _fputs fputws
#define __fsopen _wfsopen
#define _strcpy_s wcscpy_s
#define _strlen wcslen
#define _sprintf_s swprintf_s
#define _strcat_s wcscat_s
#define _strcmp wcscmp
#else
#define _fputs fputs
#define __fsopen _fsopen
#define _strcpy_s strcpy_s
#define _strlen strlen
#define _sprintf_s sprintf_s
#define _strcat_s strcat_s
```

```
#define _strcmp strcmp
#endif

/* Standard C++ headers */
#include <iostream>
#include <sstream>
#include <memory>
#include <stdexcept>

#define NA 0
#define INULL -1
#define DUMPIT 1
#define PRTINTEREST 2
#define PRTNORMAL 3
#define MONGO 4

#define useSQL 0

#define totalByDump 1 // if 0 total by appid

// Globals
using namespace std;

void strtoupperIP (CHAR *lower);
char *convert_path(char *path);
char *convert_loc(char *path);
char *convert_app(char *path);
char *null();

void getMongoValue(const bson_t *doc, const char *inputKey, void **outputValue)
    ↪ ;
char * getMongoApp(mongoc_collection_t *helixCollection, TCHAR *jobID, TCHAR *
    ↪ hostIP, TCHAR *pid);
uint8_t processMongoSample(mongoc_collection_t *collection, mongoc_collection_t
    ↪  *helixCollection, TCHAR *jobID);
```

Appendix B. Code: Whitelist

1. whitelist.cpp

```
#include "whitelist.h"

#define DBNAME "thesis"

const bson_t *currentDoc;
mongoc_client_t *client;

int _tmain(int argc, TCHAR *argv[]) {

        uint32_t i;
        int ctr;

        TCHAR wprefix[1000], wjobID[1000];

        mongoc_collection_t *collection, *processedCollection, *
            ↪ testedCollection;

        // Syntax: whitelist [load|build|test] dbname jobID(s)

        collection = NULL;
        processedCollection = NULL;
        testedCollection = NULL;

        mongoc_init ();

        client = mongoc_client_new ("mongodb://127.0.0.1/");

    //collection is the WhiteList collection. This is where Process/Modules/
        ↪ Connections are recorded when 'buildMongodb' is ran.
    //These are then used to determine WhiteList percentages when '
        ↪ testMongoSample' is used to compare against the WhiteList.
        collection = mongoc_client_get_collection (client, DBNAME, "WhiteList")
            ↪ ;
        if (collection == NULL) return 0;
        //processedCollection is the helix collection generated by Metasponse's
            ↪  helix module. This code never writes to this collection.
```

```
    processedCollection = mongoc_client_get_collection (client, DBNAME, "
        ↪ Processed");
    if (processedCollection == NULL) return 0;
    //testedCollection is the WhiteListTested collection that is used to
        ↪ track what JobID's/HostIP's have been compared against the
        ↪ WhiteList.
    testedCollection = mongoc_client_get_collection (client, DBNAME, "
        ↪ WhiteListTested");
    if (testedCollection == NULL) return 0;

    if (!_tcscmp(argv[1],"build"))
            printf("\nBuilding DB\n");
    else if (!_tcscmp(argv[1],"test"))
            printf("\nTesting sample\n");
else
printf("Using DB %s\n",DBNAME);
//Process each JobID passed to the program.
    for (ctr = 3; ctr < argc; ctr++) {
            for (i = 0; i < _tcslen(argv[ctr]); i++) wprefix[i] = argv[ctr
                ↪ ][i]; wprefix[_tcslen(argv[ctr])] = 0;
            _tcscpy(wjobID, wprefix);
            if (!_tcscmp(argv[1],"build")) {
                    if (!buildMongodb(collection, processedCollection,
                        ↪ testedCollection, wjobID)) return 0;
            } else if (!_tcscmp(argv[1],"test")) {
                    if (!testMongoSample(collection, processedCollection,
                        ↪ testedCollection, wjobID)) return 0;
            } else
                    printf("\nBad Operation Specified\nwhitelist build
                        ↪ JobID(s)\nwhitelist test JobID(s)\n");
    }
printf("Exiting\n");

//Clean up
mongoc_collection_destroy (collection);
mongoc_collection_destroy (processedCollection);
mongoc_collection_destroy (testedCollection);
    mongoc_client_destroy (client);
    mongoc_cleanup ();
```

```
        return 1;

} //END main()

uint8_t buildMongodb(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ processedCollection, mongoc_collection_t *testedCollection, TCHAR *jobID
    ↪ ) {

        int appidI, rc, numtcp, numudp;
        char *applicationS, *pathS, *str, *pidS, *parentS, *dllS, *locationS, *
            ↪ hostIP;
        bool debugit;
        const bson_t *doc, *doc2, *pipeline, *pipeline2, *query, *hostpipeline,
            ↪  *currentHost, *hostdoc;
        bson_error_t t_error;
    mongoc_cursor_t *cursor, *importCursor, *importCursor2, *hostCursor;

    //DEBUG output boolean
        debugit = false;
        t_error.domain = 0;
        t_error.code = 0;

    //Check if the jobID has already been loaded into the WhiteList
        query = BCON_NEW ( "$query", "{", "tableName", "Collections", "JobID",
            ↪ BCON_UTF8(jobID), "completed", BCON_INT32(1), "}");
        cursor = mongoc_collection_find ( collection, MONGOC_QUERY_NONE, 0, 0,
            ↪ 0, query, NULL, NULL);
        if (mongoc_cursor_next(cursor, &doc)) {
                printf("JobID Already Loaded\n");
                mongoc_cursor_destroy(cursor);
                return 1; //If the jobID has already been completed then exit
                    ↪ Build
        } else {
            printf("JobID Not Loaded\n");
        }

        //Now to check that the JobID has already been PreProcessed
        query = BCON_NEW ( "$query", "{", "tableName", "Processed", "JobID",
            ↪ BCON_UTF8(jobID), "completed", BCON_INT32(1), "}");
```

```
    cursor = mongoc_collection_find ( processedCollection ,
        ↪ MONGOC_QUERY_NONE, 0, 0, 0, query, NULL, NULL);
    if (!mongoc_cursor_next(cursor, &doc)) {
            printf("JobID has not been PreProcessed\n");
            mongoc_cursor_destroy(cursor);
            return 1;
    }

printf("Iterating through all Hosts for Job ID %s\n",jobID);

//The Host IP are iterated through after sorting. If the program ever
    ↪ crashes before completing build, then it will only lose the progress
//for the Host IP it crashed on. Re-running the same command will start
    ↪ building from the same Host IP it crashed on. (This may result in
//a handful of Processes/Modules/Connections being added twice to the
    ↪ WhiteList. Provided many Hosts are being added this should have a
//very minor impact. This impact cannot be removed without adding some
    ↪ significant overhead.

hostpipeline = BCON_NEW ( "$query", "{", "tableName", "Processed", "JobID",
    ↪  BCON_UTF8(jobID), "loaded", BCON_INT32(1), "}",
                            "$orderby", "{", "HostIP", BCON_INT32(1), "}");
    hostCursor = mongoc_collection_find ( processedCollection ,
        ↪ MONGOC_QUERY_NONE, 0, 0, 0, hostpipeline, NULL, NULL);

//Loop through each Host IP
while (mongoc_cursor_next(hostCursor, &currentHost))  {
    if (debugit) { str = bson_as_json (currentHost, NULL); printf("
        ↪ HostCursor\t\tstr = %s\n",str); bson_free (str); }
    getMongoValue(currentHost, "HostIP", (void **) &hostIP);

    //Check if the Host IP has already been loaded.
    query = BCON_NEW ( "$query", "{", "tableName", "Collections", "JobID",
        ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "loaded",
        ↪ BCON_INT32(1), "}");
    cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0, 0,
        ↪ 0, query, NULL, NULL);
    if (mongoc_cursor_next(cursor, &hostdoc)) {
        printf("Host IP %s Already Loaded\n",hostIP);
        mongoc_cursor_destroy(cursor);
```

```
        //If the Host IP has already been loaded then continue to the next
            ↪ Host IP
    } else {

        printf("Building from Process Lists for Job ID %s, Host %s\n",jobID
            ↪ ,hostIP);

        pipeline = BCON_NEW ( "$query", "{", "tableName", "PreProcessList",
            ↪  "JobID", BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
            ↪ "}");
        importCursor = mongoc_collection_find ( processedCollection,
            ↪ MONGOC_QUERY_NONE, 0, 0, 0, pipeline, NULL, NULL);

        //Iterate through each Process of the current Host IP
        while (mongoc_cursor_next(importCursor, &currentDoc))  {
            if (debugit) { str = bson_as_json (currentDoc, NULL); printf("
                ↪ ProcessList\t\tstr = %s\n",str); bson_free (str); }

            getMongoValue(currentDoc, "PID", (void **) &pidS);
            getMongoValue(currentDoc, "Application", (void **) &
                ↪ applicationS);
            getMongoValue(currentDoc, "Parent", (void **) &parentS);
            getMongoValue(currentDoc, "Location", (void **) &locationS);

            appidI = getMongoAppid(collection, testedCollection, jobID,
                ↪ hostIP, pidS, applicationS, parentS, locationS, TRUE);

            getMongoValue(currentDoc, "Path", (void **) &pathS);
            getMongoValue(currentDoc, "NumTCP", (void **) &numtcp);
            getMongoValue(currentDoc, "NumUDP", (void **) &numudp);
            if (debugit) { printf("TCP %i\tUDP %i\n",numtcp, numudp); }

            // Either add another ProcessList record or update the count on
                ↪  an existing one
            doc = BCON_NEW ("tableName", "ProcessList", "Application",
                ↪ BCON_UTF8(applicationS), "AppID", BCON_INT32(appidI),
                                "Parent", BCON_UTF8(parentS), "Path",
                                    ↪ BCON_UTF8(pathS), "Location",
                                    ↪ BCON_UTF8(locationS) );
            doc2 = BCON_NEW ("$inc", "{", "Qty", BCON_INT32(1), "}");
```

```
if (!mongoc_collection_update( collection, MONGOC_UPDATE_UPSERT
    ↪ , doc, doc2, NULL, &t_error))
    printf("Update Error: %i.%i:\t%s\n",t_error.domain,t_error.
        ↪ code,t_error.message);

doc = BCON_NEW ("tableName", "TCP", "Application", BCON_UTF8(
    ↪ applicationS), "AppID", BCON_INT32(appidI),
                    "NumTCP", BCON_INT32(numtcp) );
doc2 = BCON_NEW ("$inc", "{", "Qty", BCON_INT32(1), "}");
if (!mongoc_collection_update( collection, MONGOC_UPDATE_UPSERT
    ↪ , doc, doc2, NULL, &t_error))
    printf("Update Error: %i.%i:\t%s\n",t_error.domain,t_error.
        ↪ code,t_error.message);

doc = BCON_NEW ("tableName", "UDP", "Application", BCON_UTF8(
    ↪ applicationS), "AppID", BCON_INT32(appidI),
                    "NumUDP", BCON_INT32(numudp) );
doc2 = BCON_NEW ("$inc", "{", "Qty", BCON_INT32(1), "}");
if (!mongoc_collection_update( collection, MONGOC_UPDATE_UPSERT
    ↪ , doc, doc2, NULL, &t_error))
    printf("Update Error: %i.%i:\t%s\n",t_error.domain,t_error.
        ↪ code,t_error.message);

pipeline2 = BCON_NEW ( "$query", "{", "tableName", "PreModules
    ↪ ", "JobID", BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP
    ↪ ), "PID", BCON_UTF8(pidS), "}");
importCursor2 = mongoc_collection_find ( processedCollection,
    ↪ MONGOC_QUERY_NONE, 0, 0, 0, pipeline2, NULL, NULL);

// Inserting/ Updating Modules
while (mongoc_cursor_next(importCursor2, &currentDoc)) {
    if (debugit) { str = bson_as_json (currentDoc, NULL);
        ↪ printf("Modules\t\tstr = %s\n",str); bson_free (str)
        ↪ ; }

    getMongoValue(currentDoc, "Path", (void **) &pathS);
    getMongoValue(currentDoc, "Location", (void **) &locationS)
        ↪ ;
    getMongoValue(currentDoc, "DLL", (void **) &dllS);
```

```
                doc = BCON_NEW ("tableName", "Modules", "Application",
                    ↪ BCON_UTF8(applicationS), "AppID", BCON_INT32(appidI)
                    ↪ ,
                                    "DLL", BCON_UTF8(dllS), "Path",
                                        ↪ BCON_UTF8(pathS), "Location",
                                        ↪ BCON_UTF8(locationS));
                doc2 = BCON_NEW ("$inc", "{", "Qty", BCON_INT32(1), "}");
                if (!mongoc_collection_update( collection,
                    ↪ MONGOC_UPDATE_UPSERT, doc, doc2, NULL, &t_error))
                    printf("Update Error: %i.%i:\t%s\n",t_error.domain,
                        ↪ t_error.code,t_error.message);
            }//End Modules Loop

        }//End Processes Loop

        //Update HostIP as loaded
        doc = BCON_NEW ("tableName", "Collections", "JobID", BCON_UTF8(
            ↪ jobID), "HostIP", BCON_UTF8(hostIP), "loaded", BCON_INT32(1)
            ↪ );
        rc = mongoc_collection_insert( collection, MONGOC_INSERT_NONE, doc
            ↪ , NULL, &t_error);
        if (debugit) printf("mongoc_collection_insertrc = %i     error = %d
            ↪ .%d: %s\n",rc,t_error.domain,t_error.code,t_error.message);

        printf("Host IP %s Loaded\n", hostIP);

        mongoc_cursor_destroy(cursor);
    }//End IF for HostIP not loaded

}//End JobID Loop

//Mark the JobID as completed
doc = BCON_NEW ("tableName", "Collections", "JobID", BCON_UTF8(jobID), "
    ↪ completed", BCON_INT32(1) );
rc = mongoc_collection_insert( collection, MONGOC_INSERT_NONE, doc, NULL, &
    ↪ t_error);
if (debugit) printf("mongoc_collection_insertrc = %i     error = %d.%d: %s\
    ↪ n",rc,t_error.domain,t_error.code,t_error.message);

mongoc_cursor_destroy(importCursor);
```

```
    mongoc_cursor_destroy(importCursor2);
    mongoc_cursor_destroy(hostCursor);

    printf("\n\nBuild Finished\n\n");

        return 0;

} //END buildMongodb()

uint8_t testMongoSample(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ processedCollection, mongoc_collection_t *testedCollection, TCHAR *jobID
    ↪ ) {

        //uint32_t recCount, totCount;
        char *location, *location2;
        int appid, rc, numtcp, numudp;
        const bson_t *doc, *doc2, *pipeline, *pipeline2, *pipeline3, *
            ↪ hostpipeline, *query, *currentHost, *hostdoc, *xrefDoc, *
            ↪ moduleDoc;
        mongoc_cursor_t *importCursor, *importCursor2, *importCursor3, *
            ↪ hostCursor, *cursor;
        char *applicationS, *parentS, *dllS, *hostIP, *str;
        char *pidI;
        bool debugit = false;

        int countProcesses, countSub, countSub2, numHosts, numProcesses, numQty
            ↪ ;
        double hostScore, processScore, processWeightedScore, subScore,
            ↪ moduleScore, tcpScore, udpScore, stdevProcess, stdevHost, temp;
        stack <double> hostStack;
        stack <double> processStack;

        bson_error_t t_error;
        t_error.domain = 0;
        t_error.code = 0;

    //Check if the jobID has already been tested
        query = BCON_NEW ( "$query", "{", "tableName", "Tested", "JobID",
            ↪ BCON_UTF8(jobID), "completed", BCON_INT32(1), "}");
```

```
    cursor = mongoc_collection_find (testedCollection, MONGOC_QUERY_NONE,
        ↪ 0, 0, 0, query, NULL, NULL);
    if (mongoc_cursor_next(cursor, &doc)) {
            printf("JobID Already Loaded\n");
            mongoc_cursor_destroy(cursor);
            return 1; //If the jobID has already been completed then exit
                ↪ Build
    } else {
        if (debugit) {printf("JobID Not Loaded\n");}
    }

    //Now to check that the JobID has already been PreProcessed
    query = BCON_NEW ( "$query", "{", "tableName", "Processed", "JobID",
        ↪ BCON_UTF8(jobID), "completed", BCON_INT32(1), "}");
    cursor = mongoc_collection_find ( processedCollection,
        ↪ MONGOC_QUERY_NONE, 0, 0, 0, query, NULL, NULL);
    if (!mongoc_cursor_next(cursor, &doc)) {
            printf("JobID has not been PreProcessed\n");
            mongoc_cursor_destroy(cursor);
            return 1;
    }

//Find the total number of hosts in the WhiteList
query = BCON_NEW ("pipeline", "[",
                                        "{", "$match", "{", "tableName", "
                                            ↪ Collections", "loaded",
                                            ↪ BCON_INT32(1), "}", "}",
                                            "{", "$group", "{", "_id", "
                                                ↪ $tableName", "count",
                                                ↪ "{", "$sum", "$loaded",
                                                ↪  "}", "}", "}",
                    "]");
cursor = mongoc_collection_aggregate (collection, MONGOC_QUERY_NONE, query,
    ↪  NULL, NULL);
if (mongoc_cursor_next(cursor, &doc)) {
    getMongoValue(doc,"count", (void **) &numHosts);
}

printf("Iterating through all Hosts for Job ID %s\n",jobID);
```

```
//The Host IP are iterated through after sorting. If the program ever
    ↪ crashes before completing test, then it will only lose the progress
//for the Host IP it crashed on. Re-running the same command will start
    ↪ testing from the same Host IP it crashed on. (This may result in
//a handful of Processes/Modules/Connections being tested twice. Provided
    ↪ many Hosts are being tested, this should have a very minor impact.

hostpipeline = BCON_NEW ( "$query", "{", "tableName", "Processed", "JobID",
    ↪  BCON_UTF8(jobID), "loaded", BCON_INT32(1), "}",
                              "$orderby", "{", "HostIP", BCON_INT32(1), "}" )
                                  ↪ ;
    hostCursor = mongoc_collection_find ( processedCollection,
        ↪ MONGOC_QUERY_NONE, 0, 0, 0, hostpipeline, NULL, NULL);

//Loop through each Host IP
while (mongoc_cursor_next(hostCursor, &currentHost))  {
    if (debugit) { str = bson_as_json (currentHost, NULL); printf("Hosts\t\
        ↪ tstr = %s\n",str); bson_free (str); }
    getMongoValue(currentHost, "HostIP", (void **) &hostIP);
    hostScore = 0.0;
    stdevHost = 0.0;
    countProcesses = 0;

    query = BCON_NEW ( "$query", "{", "tableName", "Tested", "JobID",
        ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "loaded",
        ↪ BCON_INT32(1), "}");
    cursor = mongoc_collection_find (testedCollection, MONGOC_QUERY_NONE,
        ↪ 0, 0, 0, query, NULL, NULL);

    if (mongoc_cursor_next(cursor, &hostdoc)) {
        printf("Host IP %s Already Tested\n",hostIP);
        mongoc_cursor_destroy(cursor);
        //If the hostIP has already been tested then continue to the next
            ↪ host
    } else {

        printf("Testing Host IP %s\n",hostIP);
```

```
pipeline = BCON_NEW ( "$query", "{", "tableName", "PreProcessList",
    ↪  "JobID", BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
    ↪ "}");
importCursor = mongoc_collection_find ( processedCollection,
    ↪ MONGOC_QUERY_NONE, 0, 0, 0, pipeline, NULL, NULL);

//Test each Process
while ( mongoc_cursor_next(importCursor, &currentDoc) ) {
    if (debugit) { str = bson_as_json (currentDoc, NULL); printf("
        ↪ Processest\tstr = %s\n",str); bson_free (str); }
    countProcesses++;
    processScore = 0.0;
    processWeightedScore = 0.0;
    stdevProcess = 0.0;
    countSub = 0;
    countSub2 = 0;
    getMongoValue(currentDoc, "PID", (void **) &pidI);
    getMongoValue(currentDoc, "Application", (void **) &
        ↪ applicationS);
    getMongoValue(currentDoc, "Parent", (void **) &parentS);
    getMongoValue(currentDoc, "Location", (void **) &location);

    //Get the AppID of the Process in the Whitelist that is the
        ↪ nearest match to the Process being tested.
    pipeline2 = BCON_NEW ( "$query", "{", "tableName", "
        ↪ ApplicationXREF", "PID", BCON_UTF8(pidI), "JobID",
        ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "}");
    importCursor2 = mongoc_collection_find ( collection,
        ↪ MONGOC_QUERY_NONE, 0, 0, 0, pipeline2, NULL, NULL);

    if ( mongoc_cursor_next(importCursor2, &xrefDoc) ) {
        //If the PID, HostIP, and JobID match a process that was
            ↪ used to build the WhiteList, we can just grab the
            ↪ AppID from the ApplicationXREF table
        getMongoValue(xrefDoc, "AppID", (void **) &appid);
    } else {
        //Otherwise, we must do the work to find the AppID for the
            ↪ process being tested
        appid = getMongoAppid(collection,testedCollection,jobID,
            ↪ hostIP,pidI,applicationS,parentS,location,FALSE);
```

```
}
if (debugit) {printf("Processes Appid %i\n",appid);}

getMongoValue(currentDoc, "NumTCP", (void **) &numtcp);
getMongoValue(currentDoc, "NumUDP", (void **) &numudp);

if (appid == 0) {
    //The Process doesn't match any Process in the WhiteList.
    //Test TCP Connections
    doc2 = BCON_NEW ("tableName", "TCP", "JobID", BCON_UTF8(
        ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                    "PID", BCON_UTF8(pidI), "Application",
                        ↪ BCON_UTF8(applicationS), "AppID
                        ↪ ", BCON_INT32(appid), "NumTCP",
                        ↪ BCON_INT32(numtcp),
                    "TCPScore", BCON_DOUBLE(0.0) );
    rc = mongoc_collection_insert( testedCollection,
        ↪ MONGOC_INSERT_NONE, doc2, NULL, NULL);

    //Test UDP Connections
    doc2 = BCON_NEW ("tableName", "UDP", "JobID", BCON_UTF8(
        ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                    "PID", BCON_UTF8(pidI), "Application",
                        ↪ BCON_UTF8(applicationS), "AppID
                        ↪ ", BCON_INT32(appid), "NumUDP",
                        ↪ BCON_INT32(numudp),
                    "UDPScore", BCON_DOUBLE(0.0) );
    rc = mongoc_collection_insert( testedCollection,
        ↪ MONGOC_INSERT_NONE, doc2, NULL, NULL);

    //Test Modules
    pipeline2 = BCON_NEW ( "$query", "{", "tableName", "
        ↪ PreModules", "JobID", BCON_UTF8(jobID), "HostIP",
        ↪ BCON_UTF8(hostIP), "PID", BCON_UTF8(pidI), "}");
    importCursor2 = mongoc_collection_find (
        ↪ processedCollection, MONGOC_QUERY_NONE, 0, 0, 0,
        ↪ pipeline2, NULL, NULL);

    while ( mongoc_cursor_next(importCursor2, &moduleDoc) ) {
        getMongoValue(moduleDoc, "DLL", (void **) &dllS);
```

```
            getMongoValue(moduleDoc, "Location", (void **) &
                ↪ location2);

            doc2 = BCON_NEW ("tableName", "Modules", "JobID",
                ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
                                  "PID", BCON_UTF8(pidI), "
                                      ↪ Application", BCON_UTF8(
                                      ↪ applicationS), "AppID",
                                      ↪ BCON_INT32(appid),
                                  "DLL", BCON_UTF8(dllS), "Location",
                                      ↪ BCON_UTF8(location2),
                                  "ModuleScore", BCON_DOUBLE(0.0) );

            rc = mongoc_collection_insert( testedCollection,
                ↪ MONGOC_INSERT_NONE, doc2, NULL, NULL);
        }//End Modules Loop

        doc = BCON_NEW ("tableName", "ProcessList", "JobID",
            ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
                              "PID", BCON_UTF8(pidI), "AppID",
                                  ↪ BCON_INT32(appid), "Application
                                  ↪ ", BCON_UTF8(applicationS),
                              "Parent", BCON_UTF8(parentS), "Location
                                  ↪ ", BCON_UTF8(location), "
                                  ↪ ProcessScore", BCON_DOUBLE(0.0),
                              "ProcessAvg", BCON_DOUBLE(0.0), "
                                  ↪ ProcessStDev", BCON_DOUBLE(0.0),
                                  ↪  "ProcessOccurrenceCount",
                                  ↪ BCON_INT32(0) );

        rc = mongoc_collection_insert( testedCollection,
            ↪ MONGOC_INSERT_NONE, doc, NULL, NULL);

        //Update the host stack
        hostStack.push(0.0);

    } else {
        //The Application being Tested matches an Application in
            ↪ the WhiteList
        subScore = 0.0;
```

```
//Find the total number of entries for this Application in
    ↪ the WhiteList
pipeline2 = BCON_NEW ("pipeline", "[",
                        "{", "$match", "{", "tableName", "
                            ↪ ProcessList", "Application",
                            ↪  BCON_UTF8(applicationS),
                            ↪ "}", "}",
                        "{", "$group", "{", "_id",  "
                            ↪ $tableName", "count", "{", "
                            ↪ $sum", "$Qty",  "}", "}",
                            ↪ "}",
                        "]");
importCursor2 = mongoc_collection_aggregate (collection,
    ↪ MONGOC_QUERY_NONE, pipeline2, NULL, NULL);
if (mongoc_cursor_next(importCursor2, &doc)) {
    getMongoValue(doc,"count", (void **) &numProcesses);
}

//Test TCP Connections
pipeline2 = BCON_NEW ("pipeline", "[",
                        "{", "$match", "{", "tableName", "
                            ↪ TCP", "Application",
                            ↪ BCON_UTF8(applicationS), "
                            ↪ NumTCP", BCON_INT32(numtcp),
                            ↪  "}", "}",
                        "{", "$group", "{", "_id",  "
                            ↪ $tableName", "count", "{", "
                            ↪ $sum", "$Qty",  "}", "}",
                            ↪ "}",
                        "]");
importCursor2 = mongoc_collection_aggregate (collection,
    ↪ MONGOC_QUERY_NONE, pipeline2, NULL, NULL);
if (mongoc_cursor_next(importCursor2, &doc)) {
    getMongoValue(doc,"count", (void **) &numQty);
}

//Do math for TCP SubScore
if (numQty >= numProcesses) {
    tcpScore = 1.0;
```

```
} else {
    tcpScore = (float)numQty / (float)numProcesses;
}
//Stats
countSub++;
processScore += tcpScore;
processStack.push(tcpScore);
//Weighted Stats
processWeightedScore += (tcpScore * weightTCP);

doc2 = BCON_NEW ("tableName", "TCP", "JobID", BCON_UTF8(
    ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                    "PID", BCON_UTF8(pidI), "Application",
                        ↪ BCON_UTF8(applicationS), "AppID
                        ↪ ", BCON_INT32(appid), "NumTCP",
                        ↪ BCON_INT32(numtcp),
                    "TCPScore", BCON_DOUBLE(tcpScore) );
rc = mongoc_collection_insert( testedCollection,
    ↪ MONGOC_INSERT_NONE, doc2, NULL, NULL);

//Test UDP Connections
pipeline2 = BCON_NEW ("pipeline", "[",
                        "{", "$match", "{", "tableName", "
                            ↪ UDP", "Application",
                            ↪ BCON_UTF8(applicationS), "
                            ↪ NumUDP", BCON_INT32(numudp),
                            ↪  "}", "}",
                        "{", "$group", "{", "_id",  "
                            ↪ $tableName", "count", "{", "
                            ↪ $sum", "$Qty",  "}", "}",
                            ↪ "}",
                        "]");
importCursor2 = mongoc_collection_aggregate (collection,
    ↪ MONGOC_QUERY_NONE, pipeline2, NULL, NULL);
if (mongoc_cursor_next(importCursor2, &doc)) {
    getMongoValue(doc,"count", (void **) &numQty);
}

//Do math for UDP SubScore
if (numQty >= numProcesses) {
```

```
    udpScore = 1.0;
} else {
    udpScore = (float)numQty / (float)numProcesses;
}
//Stats
countSub++;
processScore += udpScore;
processStack.push(udpScore);
//Weighted Stats
processWeightedScore += (udpScore * weightUDP);

doc2 = BCON_NEW ("tableName", "UDP", "JobID", BCON_UTF8(
    ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                    "PID", BCON_UTF8(pidI), "Application",
                        ↪ BCON_UTF8(applicationS), "AppID
                        ↪ ", BCON_INT32(appid), "NumUDP",
                        ↪ BCON_INT32(numudp),
                    "UDPScore", BCON_DOUBLE(udpScore) );
rc = mongoc_collection_insert( testedCollection,
    ↪ MONGOC_INSERT_NONE, doc2, NULL, NULL);

//Test Modules
pipeline2 = BCON_NEW ( "$query", "{", "tableName", "
    ↪ PreModules", "JobID", BCON_UTF8(jobID), "HostIP",
    ↪ BCON_UTF8(hostIP), "PID", BCON_UTF8(pidI), "}");
importCursor2 = mongoc_collection_find (
    ↪ processedCollection, MONGOC_QUERY_NONE, 0, 0, 0,
    ↪ pipeline2, NULL, NULL);
subScore = 0.0;

if ( mongoc_cursor_next(importCursor2, &moduleDoc) ) {
    while ( mongoc_cursor_next(importCursor2, &moduleDoc) )
        ↪  {
        getMongoValue(moduleDoc, "DLL", (void **) &dllS);
        getMongoValue(moduleDoc, "Location", (void **) &
            ↪ location2);

        pipeline3 = BCON_NEW ("pipeline", "[",
                            "{", "$match", "{", "
                                ↪ tableName", "Modules
```

```
                                            ↪ ", "Application",
                                            ↪ BCON_UTF8(
                                            ↪ applicationS), "DLL
                                            ↪ ", BCON_UTF8(dllS),
                                            ↪ "}", "}",
                                         "{", "$group", "{", "_id",
                                            ↪  "$tableName", "
                                            ↪ count", "{", "$sum",
                                            ↪  "$Qty",  "}", "}",
                                            ↪ "}",
                                         "]");
importCursor3 = mongoc_collection_aggregate (
    ↪ collection, MONGOC_QUERY_NONE, pipeline3,
    ↪ NULL, NULL);
if (mongoc_cursor_next(importCursor3, &doc2)) {
    getMongoValue(doc2,"count", (void **) &numQty);
}

//Do math for Modules SubScore
if (numQty >= numProcesses) {
    subScore = 1.0;
} else {
    subScore = (float)numQty / (float)numProcesses;
}
countSub++;
countSub2++;
processScore += subScore;
moduleScore += subScore;
processStack.push(subScore);

doc2 = BCON_NEW ("tableName", "Modules", "JobID",
    ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP
    ↪ ),
                     "PID", BCON_UTF8(pidI), "
                        ↪ Application", BCON_UTF8(
                        ↪ applicationS), "AppID",
                        ↪ BCON_INT32(appid),
                     "DLL", BCON_UTF8(dllS), "
                        ↪ Location", BCON_UTF8(
                        ↪ location2),
```

```
                                    "ModuleScore", BCON_DOUBLE(
                                        ↪ subScore) );

            rc = mongoc_collection_insert( testedCollection ,
                ↪ MONGOC_INSERT_NONE, doc2 , NULL, NULL);
        }//End Modules Loop

        //Stats
        moduleScore = moduleScore / (float)countSub2;
        //Weighted Stats
        if (moduleScore >= 1.0) {
            moduleScore = 1.0;
            processWeightedScore += weightModule;
        } else {
            processWeightedScore += (moduleScore * weightModule
                ↪ );
        }//End Module Loop

    } else {
        moduleScore = 1.0;
        processWeightedScore += weightModule;
    }//End Modules IF

    processScore = processScore / (float)countSub; //The
        ↪ Average of all the TCP, UDP, and Module Scores

    while ( !processStack.empty() ) { //Find the variance of
        ↪ the SubScores
        temp = processStack.top();
        stdevProcess += (temp - processScore)*(temp -
            ↪ processScore);
        processStack.pop();
    }
    stdevProcess = sqrt(stdevProcess / (float)countSub); //Now
        ↪ find the Standard Deviation of the SubScores

    doc = BCON_NEW ("tableName", "ProcessList", "JobID",
        ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
```

```
                            "PID", BCON_UTF8(pidI), "AppID",
                              ↪ BCON_INT32(appid), "Application
                              ↪ ", BCON_UTF8(applicationS),
                            "Parent", BCON_UTF8(parentS), "Location
                              ↪ ", BCON_UTF8(location), "
                              ↪ ProcessScore", BCON_DOUBLE(
                              ↪ processWeightedScore),
                            "ProcessAvg", BCON_DOUBLE(processScore)
                              ↪ , "ProcessStDev", BCON_DOUBLE(
                              ↪ stdevProcess), "
                              ↪ ProcessOccurrenceCount",
                              ↪ BCON_INT32(numProcesses) );

        rc = mongoc_collection_insert( testedCollection,
            ↪ MONGOC_INSERT_NONE, doc, NULL, NULL);

        //Now to update for Scoring
        hostStack.push(processWeightedScore);
        hostScore += processWeightedScore;
    }//End AppID IF
}//End Processes Loop

//Now to score the HostIP
hostScore = hostScore / (float)countProcesses; //The Average of all
    ↪ the Process Scores
while ( !hostStack.empty() ) { //Find the variance of the
    ↪ ProcessScores
    temp = hostStack.top();
    stdevHost += (temp - hostScore)*(temp - hostScore);
    hostStack.pop();
}
stdevHost = sqrt(stdevHost / (float)countProcesses); //Now find the
    ↪ Standard Deviation of the ProcessScores

//Update HostIP as loaded
doc = BCON_NEW ("tableName", "Tested", "JobID", BCON_UTF8(jobID), "
    ↪ HostIP", BCON_UTF8(hostIP), "loaded", BCON_INT32(1),
                    "HostScore", BCON_DOUBLE(hostScore), "
                      ↪ HostStandardDeviation", BCON_DOUBLE(
                      ↪ stdevHost) );
```

```
            rc = mongoc_collection_insert(   testedCollection ,
                ↪ MONGOC_INSERT_NONE, doc , NULL, &t_error);
            if (debugit) printf(" mongoc_collection_insertrc = %i       error = %d
                ↪ .%d: %s\n",rc ,t_error.domain ,t_error.code ,t_error.message);

            printf("Host IP %s Tested\n", hostIP);

            mongoc_cursor_destroy(cursor);
        }//End IF for HostIP not loaded

    }//End JobID Loop
    mongoc_cursor_destroy(hostCursor);

    //Mark the JobID as completed
    doc = BCON_NEW ("tableName", "Tested", "JobID", BCON_UTF8(jobID), "
        ↪ completed", BCON_INT32(1) );
    rc = mongoc_collection_insert( testedCollection , MONGOC_INSERT_NONE, doc ,
        ↪ NULL, &t_error);
    if (debugit) printf(" mongoc_collection_insertrc = %i       error = %d.%d: %s\
        ↪ n",rc ,t_error.domain ,t_error.code ,t_error.message);

    if (debugit) { printf("rc = %d\n",rc); }
        return 0;

} //END testMongoSample()

/******************************************************************************/
// Convert the string to upper case and return a new string
char *strtoupper_c (const char *lower) {
        unsigned int i;
        char *upper;
        if (lower == NULL) return NULL;
        upper = (char *) malloc(sizeof(char) * (strlen(lower) + 1));
        for (i = 0; i < strlen(lower); i++)     upper[i] = toupper(lower[i]);
        upper[strlen(lower)] = 0;
        return upper;
}

// Convert the string to upper case in place (i.e., do not return a new string)
void strtoupperIP (CHAR *lower) {
```

```
        unsigned int i;
        if (lower == NULL) return;
        for (i = 0; i < strlen(lower); i++)     lower[i] = toupper(lower[i]);
}

/****************************************************************************/
// getMongoValue returns the value of the inputKey in the imported document
void getMongoValue(const bson_t *doc, const char *inputKey, void **outputValue)
    ↪ {
        char *str, *retval, *wkey, *inputKeyUpper;
        bson_iter_t iter;
        const bson_value_t *bvalue;
        int i;
        int32_t *indirectInt32;

        inputKeyUpper = strtoupper_c(inputKey);
        str = bson_as_json (doc, NULL);
        if (bson_iter_init(&iter, doc)) {
           while (bson_iter_next(&iter)) {
                   wkey = (char *) malloc(sizeof(char) * (strlen(bson_iter_key
                       ↪ (&iter)) + 1));
                   strcpy(wkey, bson_iter_key (&iter));
                   strtoupperIP(wkey);
                   if (!strcmp(wkey,inputKeyUpper)) {
                           bvalue = bson_iter_value (&iter);
                           if (bvalue->value_type == BSON_TYPE_UTF8) {
                                   retval = (char *) malloc(sizeof(char) * (
                                       ↪ bvalue->value.v_utf8.len + 1));
                                   for (i = 0; i < (int) bvalue->value.v_utf8.
                                       ↪ len; i++) retval[i] = bvalue->value.
                                       ↪ v_utf8.str[i];
                                   retval[i] = 0;
                                   free(wkey);
                                   free(inputKeyUpper);
                                   bson_free (str);
                                   *outputValue = (void *) retval;
                           } else if (bvalue->value_type == BSON_TYPE_INT32) {
                                   free(wkey);
                                   free(inputKeyUpper);
                                   bson_free (str);
```

```
                                indirectInt32 = (int32_t *) outputValue;
                                *indirectInt32 = bvalue->value.v_int32;
                        } else {
                                printf("**getValue** key = %s\t\ttype = %i\
                                    ↪ ndocument = %s\n\n",inputKey,bvalue->
                                    ↪ value_type,str);
                        }
                        return;
                }
                free(wkey);
        }
    }

        free(inputKeyUpper);
        bson_free (str);

        return;
} //END getMongoValue

// getMongoAppid uses a JobID, HostIP, and PID and determines the nearest match
    ↪ to a Process that has been assigned an AppID in the Whitelist
int getMongoAppid(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ testedCollection, TCHAR *jobID, TCHAR *hostIP, TCHAR *pid, TCHAR *
    ↪ wapplication, TCHAR *wparent, TCHAR *wlocation, bool makeChanges) {

        int rc2;
    bool appFound;
        char *appLocation, *appParent;
        uint32_t wappid, bestApp;
        uint8_t prtit;
        const bson_t *query, *doc, *doc2;
        mongoc_cursor_t *cursor;

    //DEBUG control
        prtit = 0;

        // First, check to see if we've already figured out what the appid is
        if (makeChanges) {//When makeChanges is TRUE, we check the WhiteList (
            ↪ buildMongodb)
```

```
    query = BCON_NEW ( "$query", "{", "tableName", "ApplicationXREF", "PID
        ↪ ", BCON_UTF8(pid), "JobID", BCON_UTF8(jobID), "HostIP",
        ↪ BCON_UTF8(hostIP), "}");
    cursor = mongoc_collection_find ( collection, MONGOC_QUERY_NONE, 0, 0,
        ↪ 0, query, NULL, NULL);

    if (mongoc_cursor_next(cursor, &doc)) {
        getMongoValue(doc, "AppID", (void **) &wappid);
        mongoc_cursor_destroy (cursor);
        return wappid;
    }
    } else {//When makeChanges is FALSE, we check the the testedCollection
        ↪ (testMongoSample)
    query = BCON_NEW ( "$query", "{", "tableName", "ProcessList", "PID",
        ↪ BCON_UTF8(pid), "JobID", BCON_UTF8(jobID), "HostIP", BCON_UTF8(
        ↪ hostIP), "}");
    cursor = mongoc_collection_find ( testedCollection, MONGOC_QUERY_NONE,
        ↪ 0, 0, 0, query, NULL, NULL);

    if (mongoc_cursor_next(cursor, &doc)) {
        getMongoValue(doc, "AppID", (void **) &wappid);
        mongoc_cursor_destroy (cursor);
        return wappid;
    }
    }

    appLocation = NULL;
    appFound = FALSE;
    bestApp = 0;

//Query every Whitelist process of the same name
    if (prtit) printf("pid = %s\t\twapplication = %s\n",pid,wapplication);

query = BCON_NEW ( "$query", "{", "tableName", "ProcessList", "Application
    ↪ ", BCON_UTF8(wapplication), "}");
    cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0, 0,
        ↪ 0, query, NULL, NULL);

// Iterate through all Applications of the same name in the Whitelist
    while ( (mongoc_cursor_next(cursor, &doc)) && (!appFound) ) {
```

```
    getMongoValue(doc,"Location", (void **) &appLocation);
    getMongoValue(doc,"Parent", (void **) &appParent);

    //printf("App %s\n\tLoc %s\tParent %s\nWhiteList\n\tLoc %s\tParent %s\n
        ↪ ",wapplication,wlocation,wparent,appLocation,appParent);
    if ( (!strcmp(wlocation,appLocation)?1:0) && (!strcmp(wparent,appParent
        ↪ )?1:0) ) {
        getMongoValue(doc,"AppID", (void **) &wappid);
        bestApp = wappid;
        appFound = TRUE;
    }
    }
    if (prtit) {
    if (appFound){
        printf("%s FOUND AppID %i\n",wapplication,bestApp);
    } else {
        printf("%s Not Found\n",wapplication);
    }
    }

    mongoc_cursor_destroy(cursor);

// Now that every existing Whitelist AppID has been compared, if no
    ↪ existing AppID matched then a new AppID is generated
    if ( (!appFound) && (makeChanges) ) {
            query = BCON_NEW ( "$query", "{", "tableName", "Miscellaneous",
                ↪ "fieldName", "MaxAppID", "}");
            cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE,
                ↪ 0, 0, 0, query, NULL, NULL);
            if (mongoc_cursor_next(cursor, &doc)) {
                    getMongoValue(doc,"MaxAppID",(void **) &bestApp);
        doc = BCON_NEW ("tableName", "Miscellaneous", "fieldName", "
            ↪ MaxAppID");
        doc2 = BCON_NEW ("tableName", "Miscellaneous", "fieldName", "
            ↪ MaxAppID", "MaxAppID", BCON_INT32(bestApp + 1));
                    rc2 = mongoc_collection_update(collection,
                        ↪ MONGOC_UPDATE_NONE, doc, doc2, NULL, NULL);
                    bestApp++;
            } else {
                    bestApp = 1;
```

```
            doc = BCON_NEW ("tableName", "Miscellaneous", "fieldName", "
                ↪ MaxAppID", "MaxAppID", BCON_INT32(bestApp));
                        rc2 = mongoc_collection_insert(collection,
                            ↪ MONGOC_INSERT_NONE, doc, NULL, NULL);
                }
                mongoc_cursor_destroy(cursor);
                // Get the next available Appid
                if (prtit) printf("rc2 = %i        bestApp = %i       app = %s\n",
                    ↪ rc2,bestApp,wapplication);

                // Add it into the Application table
        doc = BCON_NEW ("tableName", "Application", "AppID", BCON_INT32(bestApp
            ↪ ),
                                                    "Application", BCON_UTF8(
                                                        ↪ wapplication), "Count",
                                                        ↪  BCON_INT32(0));
                rc2 = mongoc_collection_insert(collection, MONGOC_INSERT_NONE,
                    ↪ doc, NULL, NULL);
        }

        if (makeChanges) { // i.e. getMongoAppid is being invoked from
            ↪ buildMongodb
        // A cross reference table is maintained to avoid repeating finding the
            ↪  AppID for a PID
        doc = BCON_NEW ("tableName", "ApplicationXREF", "AppID", BCON_INT32(
            ↪ bestApp), "PID", BCON_UTF8(pid), "JobID", BCON_UTF8(jobID), "
            ↪ HostIP", BCON_UTF8(hostIP));
                rc2 = mongoc_collection_insert(collection, MONGOC_INSERT_NONE,
                    ↪ doc, NULL, NULL);
        }
        return bestApp;
} //END getMongoAppid
```

2. whitelist.h

```
#include <math.h>

/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <stdint.h>
#include <stack>

#define _tmain main
#define TRUE true
#define FALSE false

#define TCHAR char
#define CHAR char

#define _tcslen strlen
#define _tcscmp strcmp
#define _tcscpy strcpy
#define _tcscat strcat
#define _totupper toupper

#include <mongoc.h>

#ifdef _UNICODE
#define _fputs fputws
#define __fsopen _wfsopen
#define _strcpy_s wcscpy_s
#define _strlen wcslen
#define _sprintf_s swprintf_s
#define _strcat_s wcscat_s
#define _strcmp wcscmp
#else
#define _fputs fputs
#define __fsopen _fsopen
#define _strcpy_s strcpy_s
#define _strlen strlen
#define _sprintf_s sprintf_s
#define _strcat_s strcat_s
```

```
#define _strcmp strcmp
#endif

/* Standard C++ headers */
#include <iostream>
#include <sstream>
#include <memory>
#include <stdexcept>

#define NA 0
#define INULL -1
#define DUMPIT 1
#define PRTINTEREST 2
#define PRTNORMAL 3
#define MONGO 4

#define useSQL 0

#define totalByDump 1 // if 0 total by appid

// These are the thresholds for how strict a Process must match to be assigned
    ↪ an existing AppID
#define weightModule 0.6
#define weightTCP 0.2
#define weightUDP 0.2

// Globals
using namespace std;

void strtoupperIP (CHAR *lower);

void getMongoValue(const bson_t *doc, const char *inputKey, void **outputValue)
    ↪ ;
int getMongoAppid(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ testedCollection, TCHAR *jobID, TCHAR *hostIP, TCHAR *pid, TCHAR *
    ↪ wapplication, TCHAR *wparent, TCHAR *wlocation, bool makeChanges);
uint8_t buildMongodb(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ helixCollection, mongoc_collection_t *testedCollection, TCHAR *jobID);
uint8_t testMongoSample(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ helixCollection, mongoc_collection_t *testedCollection, TCHAR *jobID);
```

Appendix C. Code: Results Generation

1. results.cpp

```
#include "results.h"

#define DBNAME "thesis"

//These options will limit the number of results returned.
#define resultHosts 50
#define resultProcs 100

const bson_t *currentDoc;
mongoc_client_t *client;

int _tmain(int argc, TCHAR *argv[]) {

        uint32_t i;
        int ctr;

        TCHAR wprefix[1000], wjobID[1000];

        mongoc_collection_t *collection, *resultsCollection;

        // Syntax: whitelist [process] dbname jobID(s)

        collection = NULL;
        resultsCollection = NULL;

        mongoc_init ();

        client = mongoc_client_new ("mongodb://127.0.0.1/");

    //collection is the PreProcessed collection. This is where Process/Modules/
        ↪ Connections are processed from the helix collection
    //into simple key/value pairs to speed up the processing during build and
        ↪ test operations.
        collection = mongoc_client_get_collection (client, DBNAME, "
            ↪ WhiteListTested");
        if (collection == NULL) return 0;
```

```
        //helixCollection is the helix collection generated by Metasponse's
            ↪ helix module. This code never writes to this collection.
        resultsCollection = mongoc_client_get_collection (client, DBNAME, "
            ↪ WhiteListResults");
        if (resultsCollection == NULL) return 0;

        if (!_tcscmp(argv[1],"results"))
                printf("\nGenerating Results\n");

    printf("Using Database %s\n",DBNAME);

    //Process each JobID passed to the program.
        for (ctr = 3; ctr < argc; ctr++) {
                for (i = 0; i < _tcslen(argv[ctr]); i++) wprefix[i] = argv[ctr
                    ↪ ][i]; wprefix[_tcslen(argv[ctr])] = 0;
                _tcscpy(wjobID, wprefix);
                if (!_tcscmp(argv[1],"results")) {
                        if (!results(collection, resultsCollection, wjobID))
                            ↪ return 0;
                } else
                        printf("\nBad Operation Specified\nwhitelist results
                            ↪ database JobID(s)\n");
        }
    printf("Exiting\n");

    //Clean up
    mongoc_collection_destroy (collection);
    mongoc_collection_destroy (resultsCollection);
        mongoc_client_destroy (client);
        mongoc_cleanup ();

        return 1;

} //END main()

uint8_t results(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ resultsCollection, TCHAR *jobID) {

        const bson_t *doc, *hostpipeline, *pipeline, *query, *currentHost, *
            ↪ subDoc;
```

```
    mongoc_cursor_t *hostCursor, *cursor, *importCursor;
    char *hostIP, *str, *applicationS, *parentS, *locationS, *pidS, *dllS,
        ↪ *dllLocS;
    int rc, appID, processCount, i, j, numTCP, numUDP;
    double hostScore, hostStDev, processScore, processStDev, tcpScore,
        ↪ udpScore, subScore;
    bool debugit = false;

    bson_error_t t_error;
    t_error.domain = 0;
    t_error.code = 0;

//Check if the jobID has already been processed
    query = BCON_NEW ( "$query", "{", "tableName", "Tested", "JobID",
        ↪ BCON_UTF8(jobID), "completed", BCON_INT32(1), "}");
    cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0, 0,
        ↪ 0, query, NULL, NULL);
    if ( !mongoc_cursor_next(cursor, &doc) ) {
            printf("JobID hasn't been Tested\n");
            mongoc_cursor_destroy(cursor);
            return 1; //If the jobID has already been processed then exit
    } else {
        if (debugit) {printf("JobID has been Tested\n");}
    }

printf("Iterating through all Hosts for Job ID %s\n",jobID);

//The Host IP are iterated through after sorting. If the program ever
    ↪ crashes before completing test, then it will only lose the progress
//for the Host IP it crashed on. Re-running the same command will start
    ↪ testing from the same Host IP it crashed on. (This may result in
//a handful of Processes/Modules/Connections being processed twice.
    ↪ Provided many Hosts are being processed, this should have a very
    ↪ minor impact.

hostpipeline = BCON_NEW ("pipeline", "[",
                    "{", "$match", "{", "tableName", "Tested", "JobID",
                        ↪ BCON_UTF8(jobID), "loaded", BCON_INT32(1), "}",
                        ↪ "}",
```

```
                    "{", "$sort", "{", "HostScore", BCON_INT32(1), "}",
                        ↪ "}",
                    "{", "$limit", BCON_INT32(resultHosts), "}",
                    "]");

hostCursor = mongoc_collection_aggregate (collection,MONGOC_QUERY_NONE,
    ↪ hostpipeline,NULL,NULL);

if (mongoc_cursor_error(hostCursor,&t_error))
    printf("\tmongoc_collection_aggregate    error = %d.%d: %s\n",t_error.
        ↪ domain,t_error.code,t_error.message);

i = 0;
//Loop through each Host IP
while (mongoc_cursor_next(hostCursor, &currentHost)) {
    if (debugit) { str = bson_as_json (currentDoc, NULL); printf("Hosts\t\
        ↪ tstr = %s\n",str); bson_free (str); }

    i++;

    getMongoValue(currentHost, "HostIP", (void **) &hostIP);
    getMongoValue(currentHost, "HostScore", (void **) &hostScore);
    getMongoValue(currentHost, "HostStandardDeviation", (void **) &
        ↪ hostStDev);

    doc = BCON_NEW ("tableName", "CollectionResults", "JobID", BCON_UTF8(
        ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                    "HostRank", BCON_INT32(i),
                    "HostScore", BCON_DOUBLE(hostScore), "
                        ↪ HostStandardDeviation", BCON_DOUBLE(
                        ↪ hostStDev)
                );

    rc = mongoc_collection_insert( resultsCollection, MONGOC_INSERT_NONE,
        ↪ doc, NULL, NULL);

    j = 0;

    pipeline = BCON_NEW ("pipeline", "[",
```

```
                    "{", "$match", "{", "tableName", "ProcessList", "
                       ↪ JobID", BCON_UTF8(jobID), "HostIP",
                       ↪ BCON_UTF8(hostIP), "}", "}",
                    "{", "$sort", "{", "ProcessScore", BCON_INT32(1),
                       ↪ "}", "}",
                    "{", "$limit", BCON_INT32(resultProcs), "}",
                    "]");

      importCursor = mongoc_collection_aggregate (collection,
         ↪ MONGOC_QUERY_NONE, pipeline ,NULL,NULL);

      if (mongoc_cursor_error(importCursor,&t_error))
         printf("\tmongoc_collection_aggregate    error = %d.%d: %s\n",
            ↪ t_error.domain,t_error.code,t_error.message);

      //Iterate through each Process
      while ( mongoc_cursor_next(importCursor, &currentDoc) ) {
         if (debugit) { str = bson_as_json (currentDoc, NULL); printf("
            ↪ Processes\tstr = %s\n",str); bson_free (str); }
         j++;

         getMongoValue(currentDoc, "PID", (void **) &pidS);
         getMongoValue(currentDoc, "AppID", (void **) &appID);
         getMongoValue(currentDoc, "Application", (void **) &applicationS);
         getMongoValue(currentDoc, "Parent", (void **) &parentS);
         getMongoValue(currentDoc, "Location", (void **) &locationS);
         getMongoValue(currentDoc, "ProcessScore", (void **) &processScore);
         getMongoValue(currentDoc, "ProcessStDev", (void **) &processStDev);
         getMongoValue(currentDoc, "ProcessOccurrenceCount", (void **) &
            ↪ processCount);

         query = BCON_NEW ( "$query", "{", "tableName", "TCP", "JobID",
            ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "PID",
            ↪ BCON_UTF8(pidS), "}");
         cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0,
            ↪ 0, 0, query, NULL, NULL);

         while ( mongoc_cursor_next(cursor, &subDoc) ) {
            getMongoValue(subDoc, "NumTCP", (void **) &numTCP);
            getMongoValue(subDoc, "TCPScore", (void **) &tcpScore);
```

```
}

query = BCON_NEW ( "$query", "{", "tableName", "UDP", "JobID",
    ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "PID",
    ↪ BCON_UTF8(pidS), "}");
cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0,
    ↪ 0, 0, query, NULL, NULL);

while ( mongoc_cursor_next(cursor, &subDoc) ) {
    getMongoValue(subDoc, "NumUDP", (void **) &numUDP);
    getMongoValue(subDoc, "UDPScore", (void **) &udpScore);
}

doc = BCON_NEW ("tableName", "ProcessResults", "JobID", BCON_UTF8(
    ↪ jobID), "HostIP", BCON_UTF8(hostIP),
                    "HostRank", BCON_INT32(i), "ProcessRank",
                        ↪ BCON_INT32(j), "AppID", BCON_INT32(appID
                        ↪ ),
                    "PID", BCON_UTF8(pidS), "Application",
                        ↪ BCON_UTF8(applicationS),
                    "Parent", BCON_UTF8(parentS), "Location",
                        ↪ BCON_UTF8(locationS),
                    "ProcessScore", BCON_DOUBLE(processScore), "
                        ↪ ProcessStDev", BCON_DOUBLE(processStDev)
                        ↪ , "ProcessOccurrenceCount", BCON_DOUBLE(
                        ↪ processCount),
                    "NumTCP", BCON_INT32(numTCP), "TCPScore",
                        ↪ BCON_DOUBLE(tcpScore), "NumUDP",
                        ↪ BCON_INT32(numUDP), "UDPScore",
                        ↪ BCON_DOUBLE(udpScore)
                );

rc = mongoc_collection_insert( resultsCollection,
    ↪ MONGOC_INSERT_NONE, doc, NULL, NULL);

query = BCON_NEW ( "$query", "{", "tableName", "Modules", "JobID",
    ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP), "PID",
    ↪ BCON_UTF8(pidS), "}");
cursor = mongoc_collection_find (collection, MONGOC_QUERY_NONE, 0,
    ↪ 0, 0, query, NULL, NULL);
```

```
            while ( mongoc_cursor_next(cursor, &subDoc) )  {
                getMongoValue(subDoc, "DLL", (void **) &dllS);
                getMongoValue(subDoc, "Location", (void **) &dllLocS);
                getMongoValue(subDoc, "ModuleScore", (void **) &subScore);

                doc = BCON_NEW ("tableName", "ModuleResults", "JobID",
                    ↪ BCON_UTF8(jobID), "HostIP", BCON_UTF8(hostIP),
                                "HostRank", BCON_INT32(i), "ProcessRank",
                                    ↪ BCON_INT32(j), "AppID", BCON_INT32(appID
                                    ↪ ),
                                "PID", BCON_UTF8(pidS), "Application",
                                    ↪ BCON_UTF8(applicationS),
                                "DLL", BCON_UTF8(dllS), "Location", BCON_UTF8(
                                    ↪ dllLocS),
                                "ModuleScore", BCON_DOUBLE(subScore)
                            );

                rc = mongoc_collection_insert( resultsCollection,
                    ↪ MONGOC_INSERT_NONE, doc, NULL, NULL);
            }
        }
        printf("Host %s Completed\n",hostIP);
    }//End JobID Loop
    mongoc_cursor_destroy(importCursor);
    mongoc_cursor_destroy(cursor);
    mongoc_cursor_destroy(hostCursor);

    if (debugit) { printf("rc = %d\n",rc); }
        return 0;

} //END processMongoSample()

/*****************************************************************************/
// Convert the string to upper case and return a new string
char *strtoupper_c (const char *lower) {
        unsigned int i;
        char *upper;
        if (lower == NULL) return NULL;
        upper = (char *) malloc(sizeof(char) * (strlen(lower) + 1));
```

```
        for (i = 0; i < strlen(lower); i++)      upper[i] = toupper(lower[i]);
        upper[strlen(lower)] = 0;
        return upper;
}

// Convert the string to upper case in place (i.e., do not return a new string)
void strtoupperIP (CHAR *lower) {
        unsigned int i;
        if (lower == NULL) return;
        for (i = 0; i < strlen(lower); i++)      lower[i] = toupper(lower[i]);
}

/***************************************************************************/
// getMongoValue returns the value of the inputKey in the imported document
void getMongoValue(const bson_t *doc, const char *inputKey, void **outputValue)
    ↪ {
        char *str, *retval, *wkey, *inputKeyUpper;
        bson_iter_t iter;
        const bson_value_t *bvalue;
        int i;
        int32_t *indirectInt32;
        double *indirectDouble;

        inputKeyUpper = strtoupper_c(inputKey);
        str = bson_as_json (doc, NULL);
        if (bson_iter_init(&iter, doc)) {
           while (bson_iter_next(&iter)) {
                   wkey = (char *) malloc(sizeof(char) * (strlen(bson_iter_key
                       ↪ (&iter)) + 1));
                   strcpy(wkey, bson_iter_key (&iter));
                   strtoupperIP(wkey);
                   if (!strcmp(wkey,inputKeyUpper)) {
                           bvalue = bson_iter_value (&iter);
                           if (bvalue->value_type == BSON_TYPE_UTF8) {
                                   retval = (char *) malloc(sizeof(char) * (
                                       ↪ bvalue->value.v_utf8.len + 1));
                                   for (i = 0; i < (int) bvalue->value.v_utf8.
                                       ↪ len; i++) retval[i] = bvalue->value.
                                       ↪ v_utf8.str[i];
                                   retval[i] = 0;
```

```
                                        free(wkey);
                                        free(inputKeyUpper);
                                        bson_free (str);
                                        *outputValue = (void *) retval;
                                } else if (bvalue->value_type == BSON_TYPE_INT32) {
                                        free(wkey);
                                        free(inputKeyUpper);
                                        bson_free (str);
                                        indirectInt32 = (int32_t *) outputValue;
                                        *indirectInt32 = bvalue->value.v_int32;
                                } else if (bvalue->value_type == BSON_TYPE_DOUBLE) {
                                        free(wkey);
                                        free(inputKeyUpper);
                                        bson_free (str);
                                        indirectDouble = (double *) outputValue;
                                        *indirectDouble = bvalue->value.v_double;
                                } else {
                                        printf("**getValue** key = %s\t\ttype = %i\
                                            ↪ ndocument = %s\n\n",inputKey,bvalue->
                                            ↪ value_type,str);
                                }
                                return;
                        }
                        free(wkey);
                }
        }

        free(inputKeyUpper);
        bson_free (str);

        return;
} //END getMongoValue
```

2. results.h

```
#include <math.h>

/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <stdint.h>
#include <string.h>

#define _tmain main
#define TRUE true
#define FALSE false

#define TCHAR char
#define CHAR char

#define _tcslen strlen
#define _tcscmp strcmp
#define _tcscpy strcpy
#define _tcscat strcat
#define _totupper toupper

#include <mongoc.h>

#ifdef _UNICODE
#define _fputs fputws
#define __fsopen _wfsopen
#define _strcpy_s wcscpy_s
#define _strlen wcslen
#define _sprintf_s swprintf_s
#define _strcat_s wcscat_s
#define _strcmp wcscmp
#else
#define _fputs fputs
#define __fsopen _fsopen
#define _strcpy_s strcpy_s
#define _strlen strlen
#define _sprintf_s sprintf_s
#define _strcat_s strcat_s
```

```
#define _strcmp strcmp
#endif

/* Standard C++ headers */
#include <iostream>
#include <sstream>
#include <memory>
#include <stdexcept>

#define NA 0
#define INULL -1
#define DUMPIT 1
#define PRTINTEREST 2
#define PRTNORMAL 3
#define MONGO 4

#define useSQL 0

#define totalByDump 1 // if 0 total by appid

// Globals
using namespace std;

void strtoupperIP (CHAR *lower);

void getMongoValue(const bson_t *doc, const char *inputKey, void **outputValue)
    ↪ ;
uint8_t results(mongoc_collection_t *collection, mongoc_collection_t *
    ↪ resultsCollection, TCHAR *jobID);
```

Appendix D. Results Extraction Scripts

1. ResultsProcessList.sh

```
echo "Exporting Processes"
mongoexport -d thesis -c WhiteListResults -f JobID,HostIP
    ↪ ,HostRank,ProcessRank,AppID,PID,Application,Parent,
    ↪ Location,ProcessScore,ProcessStDev,
    ↪ ProcessOccurrenceCount,NumTCP,TCPScore,NumUDP,
    ↪ UDPScore -q '{"tableName":"ProcessResults"}' --csv
    ↪ -o ResultsProcesses.csv
```

2. ResultsModules.sh

```
echo "Exporting Modules"
mongoexport -d thesis -c WhiteListResults -f JobID,HostIP
    ↪ ,HostRank,ProcessRank,AppID,PID,Application,DLL,
    ↪ Location,ModuleScore -q '{"tableName":"
    ↪ ModuleResults"}' --csv -o ResultsModules.csv
```

3. ResultsCollections.sh

```
echo "Exporting Collection Info"
mongoexport -d thesis -c WhiteListResults -f JobID,HostIP
    ↪ ,HostRank,HostScore,HostStandardDeviation -q '{"
    ↪ tableName":"CollectionResults"}' --csv -o
    ↪ ResultsCollection.csv
```

Bibliography

1. F. Freiling. A Common Process Model for Incident Response and Computer Forensics. Technical report, Laboratory for Dependable Distributed Systems, University of Mannheim, Germany, 2007.

2. S. Garfinkel. Digital Forensics. Last Accessed: Feb 20, 2016, December 2013. *http://www.americanscientist.org/issues/pub/digital-forensics*.

3. W. Lynn. DoD Executive Agent for the DoD Cyber Crime Center. Last Accessed: Feb 20, 2016, March 2010. *http://www.dtic.mil/whs/directives/corres/pdf/550513E.pdf*.

4. FBI Regional Computer Forensics Laboratory Annual Report, 2012.

5. J. Clark N. Beebe. Dealing with Terabyte Data Sets in Digital Investigations. In *Advances in Digital Forensics, IFIP International Conference on Digital Forensics*, volume 194 2005, pages 3–16. The International Federation for Information Processing, February 2005.

6. LTC C. Thomas. Human Resources Command stands up Cyber Branch. Last Accessed Feb 19, 2016, 2014. *http://www.army.mil/article/122456/Human_Resources_Command_stands_up _Cyber_Branch/*.

7. F. Breitinger and H.Baier. Performance Issues About Context Triggered Piecewise Hashing. In *Digital Forensics and Cyber Crime: Third International ICST Conference*, volume Pavel Gladyshev, Marcus K. Rogers, pages 141–155. Springer, October 2011.

8. J. Spring T. Shimeall. *Introduction to Information Security: A Strategic-Based Approach*. Syngress Publishing, 14th edition, 2014.

9. D. Shackleford. Application Whitelisting: Enhancing Host Security. *SANS Whitepaper*, October 2009.

10. J. Pelzl C. Paar. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer, 2010.

11. C. Winter H. Baier A. Rybalchenko M. Steinebach F. Breitinger, H. Liu. Towards a Process Model for Hash Functions in Digital Forensics. In *Digital Forensics and Cyber Crime: Fifth International ICST Conference*, volume 132, pages 170–186. Springer, December 2014.

12. Sudarshan S. Chawathe. Effective Whitelisting for Filesystem Forensics. In *Intelligence and Security Informatics, IEEE International Conference*, pages 131–136, June 2009.

13. H. Baier C. Busch F. Breitinger, K. P. Astebol. mvHash-B: A New Approach for Similarity Preserving Hashing. In *IT Security Incident Management and IT Forensics (IMF), 2013 Seventh International Conference*, pages 33–44. Biometrics and Internet Security Research Group, Hochschule Darmstadt, IEEE, March 2013.

14. e fense. Helix3 Pro. Last Accessed: Feb 22, 2016, 2014. *http://www.e-fense.com/helix3pro.php*.

15. MongoDB, Inc. *MongoDB Documentation: Release 2.4.14*, October 2015.

16. Air Force Institute of Technology J. Okolica. Personal interview, October 2015.

17. Clymb3r. Intercepting Password Changes with Function Hooking. Last Accessed: Feb 20, 2016, September 2013. *https://clymb3r.wordpress.com/2013/09/15/intercepting-password-changes-with-function-hooking/*.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2016	Master's Thesis	Aug 2014 — Mar 2016

4. TITLE AND SUBTITLE

Statistic Whitelisting for Enterprise Network Incident Response

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Grunzweig, Nathan E., CPT, USA

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-16-M-019

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

This research seeks to satisfy the need for the rapid evaluation of enterprise network hosts in order to identify items of significance through the introduction of a statistic whitelist based on the behavior of the processes on each host. By taking advantage of the repetition of processes and the resources they access, a whitelist can be generated using large quantities of host machines. For each process, the Modules and the TCP & UDP Connections are compared to identify which resources are most commonly accessed by each process. Results show 47% of processes receiving a whitelist score of 75% or greater in the five hosts identified as having the worst overall scores and 60% of processes when the hosts more closely match the hosts used to build the whitelist.

15. SUBJECT TERMS

Digital Forensics, Whitelist

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson, AFIT/ENG
U	U	U	U	106	19b. TELEPHONE NUMBER *(include area code)* (937) 255-6565, x4281; gilbert.peterson@afit.edu